НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050201 «Системна інженерія»

на тему: «Система відображення та аналізу даних технологічного процесу виготовлення шоколаду»

Виконала:
студентка IV курсу, групи ІА-51
Вознюк Оксана Олегівна _____

Керівник:
Старший викладач кафедри АУТС
Шимкович В.М.
_____

Рецензент:
Доцент кафедри ТК, кандидат технічних наук
доцент, Тимошин Ю.А. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 рік

Пояснювальна записка
до дипломного проекту
на тему: «Система відображення та аналізу даних технологічного процесу виготовлення шоколаду»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

IT – Інформаційні Технології

CSV – Coma Separated Values

SCADA – Supervisory Control And Data Acquisition

OPC – Open Platform Communications

IDE – Integrated Development Environment

IEEE – Institute of Electrical and Electronics Engineers

JVM – Java Virtual Machine

JIT – Just In Time

HTTP – Hypertext transfer protocol

JRE – Java Runtime Environment

LIFO – Last In First Out

SOLID – Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion

ВСТУП

Єтап історії людства з відходом від ручної праці та переходом до індустріалізації можна вважати початком стрімкого розвитку виробництва. З того часу воно відігравало основну роль в забезпеченні людини матеріальними благами, його продукти всюди супроводжують людину і на сьогоднішній день.

Виробництво зазнавало значних змін – з плином часу був пройдений довгий шлях від механічних приладів, що повністю залежали від людини до сучасних автоматизованих систем, що використовують нейронні мережі, штучний інтелект, супутниковий зв'язок та багато іншого. Це призвело до того, що сучасний технологічний процес – складна система, що об'єднує велику кількість внутрішніх підсистем, сервісів, програмного забезпечення, тощо. В той самий час, посвякденне життя сучасної людини все більше залежить від результатів виробництва, тому питання його якісної підтримки, модернізації, уникнення збоїв та аварійних ситуацій набуває ще більшої значущості.

Об'єктом дослідження даного проекту виступають системи автоматизації технологічного процесу виготовлення шоколаду. Технологічні операції даного процесу дуже варіативні, оскільки практично всі показники приладів, інгредієнти та процеси обробки залежать від обраного рецепту. Використання систем автоматизації технологічного процесу суттєво спрощує управління процесом, допомагає зробити виробництво більш продуктивним та безпечним.

Процес виготовлення шоколаду має велику складність, оскільки процес обробки сировини проходить багато технологічних операцій, які мають підтримувати вивірені показники температури, вологості, тиску та інші. Невідповідність будь-якого з заданих показників встановленим нормам невідворотно псує продукцію – наприклад, якщо в результаті збою на виробництві шоколадна маса буде оброблюватись при нижчій температурі, смак шоколаду буде слабший, а поверхню плитки вкриє білий наліт, тобто зіпсується і смак і товарний вигляд, а це негативно вплине на репутацію виробництва.

Заходи запобіжності від таких наслідків формують необхідність

впроваження окремої підсистеми в технологічному процесі, яка відповідала б за спостереження за параметрами технологічного процесу. Її практична користь може бути виражена вчасним повідомленням про несправності підсистем, приладів, наданням додаткової інформації для більш швидкої та точної діагностики збою, зручним представленням обробленої інформації, тощо.

Невід'ємність такого процесу породила низку архітектурних підходів проектування систем, що забезпечують збір внутрішньої інформації технологічного процесу, її збереження та аналіз. Такий підхід широко використовується не тільки в сфері виробництва. Наприклад, звичайною практикою проектування програмного забезпечення, що має має зв'язок з сервером, виступає формування та збереження запитів (як користувацьких так і внутрішніх), технічної інформації та метаданних на сервері в окремих файлах, часто з росширенням .log, тому процес називають «логуванням», а збереженні дані – «логами» . В технологічних процесах на виробницві такі системи також запроваджуються. Проте більшість рішень, представлених зараз на ринку, об'єднує ряд недоліків, таких як:

- рішення представляє собою окрему слідкуючу систему, яка часто не може повністю інтегруватись в даний технологічний процес або не може підтримувати його повністю через складність та спецефічність виробництва;
- рішення дуже масштабне, але його можливості більше акцентовані на менеджмент та слідкування за загальними характеристиками технологічного процесу;
- висока вартість вартість продукту та його впровадження.

Наведені вище недоліки існуючих рішень та той факт, що сучасне виробництво шоколаду не має впровадженої практики встановлення власної подібної системи обумовлюють актуальність розробки системи відображення та аналізу внутрішньої інформації технологічного процесу виготовлення шоколаду.

Метою даного проекту є проектування та розробка автоматизованої системи відображення та аналізу даних технологічного процесу виготовлення шоколаду.

Її впровадження дозволить працівникам на виробництві ефективніше упрявляти технологічним процесом, відображення параметрів зробить його прозорішим, а в разі винекнення несправності допоможе швидше локалізувати несправність.

Завдання, що були визначені на дипломний проект наступні:

- аналіз предметної області та існуючих рішень представлених на ринку. Визначення ступені їх відповідності до використання на виробництві шоколаду;
- за результатами аналізу переваг та недоліків наявних рішень формування вимог до розроблюваної системи;
- визначення технічних засобів розробки системи, методів аналізу та відображення інформації;
- розробка архітектури системи, логіки її роботи, реалізація обраних алгоритмів відображення та аналізу даних;
- побудова та опис дігарам, що зображують алгоритм роботи системи та її структуру.

Практичне значення розробленої системи пролягає в наданні прозорості протікання технологічних операцій на виробництві та первинного аналізу даних технологічного процесу за допомогою зручного користувацього інтерфейсу.

Даний проект має наступну структуру:

- вступ;
- дослідження предметної області;
- технології для розробки;
- архітектура розробленої системи;
- висновки;
- список використаних джерел;
- ілюстративний матеріал.

Графічні додатки включають в себе чотири діаграми – структурну схема системи, функціональна сема, схема алгоритму аналізу даних та схема .які описують принцип роботи розробленої системи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

1.1.1 Опис технологічного процесу виготовлення шоколаду

Сучасний технологічний процес виготовлення шоколадих виробів є дуже варіативний, оскільки кінцевий продукт залежить як від домішкових інгредієнтів так і від параметрів їх обробки. Технологію виготовлення чорного шоколаду можна виділити наступні єтапи виробництва:

- обробка сировини (какао бобів, цукру, тощо) та змішування компонентів;
- додаткове перемелення та розрідження;
- тривалий процес підігріву та перемішування шоколадних мас;

Розглянемо кожний з етапів детальніше. Основний інгредієнт шоколадних мас – какао боби. Їх видобувають з плодів какао-дерева, а за структурою вони складаються з ядра та твердої оболонки - какаовели. В сирому вигляді какао боби мають кисло-гіркий смак, тому їх первинна обробка включає зменшення вмісту вологи ядрер до 2-3%, відділення ядра від какаовели, обсмаження та подрібнення до шматочків розміром 7.5-8 мм. Частина крупки використовується для видобування олії какао шляхом перемелення при високій температурі. В залежності від рецепту, цукор може попередньо промивається та також подрібнюється.

Оскільки повний процес обробки сировини тривалий та кропіткий (у випадку перемелення цурку – процес навіть вибухонебезпечний), на багатьох підприємствах замовляють необхідного розміру какао крупку та цукрову пудру.

В залежності від типу змішуючого обладнання можна виділити два підходи – неперервне та порціонне змішування. Міксери неперервної дії, в порівнянні з порціонними, більш потужні та економічно вигідні. Невеликі помилки в об'ємах основних змішуваних інгредієнтів – крупки, олії какао та цукрової пудри практично не вплинуть на якість кінцевого продукту. Проте навіть невеликі помилки в об'ємах доданих ємульгаторів та ароматизаторів будуть суттєвими та приведуть до псування цілої партії шоколаду. Тому міксери

порціоної дії вважаються більш надійними.

На виході з міксеру отримується шоколадна маса, вагомим показником якої є в'язкість. Вона поступає на подрібнювальний аппарат. В основі його конструкції полягають послідовно розташовані циліндри, що обертаючись щільно притиснені між собою. Тому, якщо шоколадна маса буде занадто густою, вона буде нерівномірно поступати на циліндри, а занадто рідка приведе їх перегріву. На цьій стадії суміш набуває сипучого стану, розмір окремих часток якої не перевищує 35 мкм, та розріджується олією какао та іншими природними розріджувачами (наприклад, пальмовою олією).[1]

Важливо зберегти пропорцію вмісту олії в кінцевій масі - 30-36% забезпечують оптимальну розрідженність при формуванні шоколаду. Розріджена шоколадна маса подається на віхд конш-пристрою.

Конширування – спецефічний процес обробки шоколадних мас, який був винайдений Рудольфом Ліндтом в 1879 році. Він полягає в неперервному перемішуванні суміші при постійній температурі 55-75°С та безпосереднім контактом з повітрям. Ця процедура є доволі тривалою – процес може протікати від 8 годин до 5 діб в окремих випадках. Після його завершення готові шоколадні маси охолоджують до температури 40-45°С та відправляють на збереження (в контейнерах з нержавіючої сталі повинна підтримуватись задана температура) та формування готового виробу.

В залежності від рецепту шоколаду, кількість інгредієнтів, їх вміст та параметри технологічного процесу можуть змінюватись. Наприклад, при виготовленні молочного шоколаду, на етапі попередньої обробки сировини, до какао-крупки додається сухе молоко або молочна сироватка. Також, подріблення суміші до розміру окремих часток 35мкм вже не необхідне – 65мкм буде достатньо, а деякі виробники змелюють і до 75мкм. Конширування молочного шоколаду займає менше часу (процес, серед іншого, використовується для позбавлення від залишків дубильних речовин, які переважно містяться в какао бобах; зменшивши їх вміст в суміші необхідна тривалість процесу зменшується) та потребує нижчих температур.

Серед великої кількості шоколадних виробів є сенс розглядати саме технологічний процес виготовлення темного шоколаду, оскільки його технологічні операції в більшій чи меншій степені модифікуючи повторюють виробництва інших шоколадних продуктів. Це робить виробництво темного шоколаду базовим та універсальним.

Проаналізувавши технологічний процес можна розділити параметрами, якими він оперує на дві групи – показники, які критично вплинуть на якість продукці, при їх невідповідності рецепту або технологічним нормам, та менш суттєві параметри.

До менш суттєвих показників відносяться:

- енергопостачання окремих приладів;
- об’єм основної сировини;
- швидкість роботи міксерів, тощо.

Звісно, ці показники залишаються важливими, проте, при невеликих відхиленнях від норми, кінцевий виріб практично не зазнає змін.

До більш критичних параметрів відносяться:

- об’єм емульгаторів та ароматизаторів;
- в’язкість суміші на кожному етапі технологічного процесу;
- ступінь подріблення суміші;
- температура суміщі всередині коншмашини та температура збереження готової шоколадної маси;
- температури окремих елементів приладів (наприклад, валів подрібнючих пристроїв);
- температура та відносна вологість в приміщені.

Невідповідність наведених вище показників може привести до псування цілої партії шоколаду, та навіть до аварійних випадків на виробництві. Тому система відображення та аналізу внутрішньої інформації технологічного процесу виготовлення шоколаду має в першу чергу орієнтуватись саме на ці показники, але також оброблювати й менш суттєві, враховуючи певну похибку.

1.1.2 Опис процесу логування

Ідея логування прийшла в сферу виробництва зі сфери Інформаційних тенологій (IT). Там вони використовуються в процесах розробки та підтримки програмного забезпечення та в процесі адміністрування мереж.

Перший етап процесу логування полягає в виборі подій, про які необхідно зберегти інформацію. Для цього потрібно визначити за якою метою буде відбуватись збір інформації. Мета часто специфічна. Наприклад, в області веб-застосунків аналіз збережених даних виявиться корисним при розслідуванні технічних несправностей, атак злоумисників, недоступності додатку або збоїв його функціоналу. Також, розробники часто впроваджують логування для аналізу дій користувача з точки зору маркетології. Визначення необхідної інформації допомагає зменшити кількість подій що буде підлягати збереженню. Це розповсюджена практика, оскільки збереження всіх операцій системи має ряд суттєвих недоліків:

- при завантаженості системи дані можуть записуватись непослідовно, хаотично. Це зробить лог складним для читання та аналізу;
- постійне звернення до файлу логу та запис в нього сильно знизить продуктивність системи.

Коли обрана цільова інформація та події, дані яких будуть збережені, необхідно визначити в якому вигляді буде відбуватись збереження даних. Перш за все це тип файлу. Зараз розповсюджені наступні формати представлення даних:

- текстовий файл;
- XML файл;
- CSV файл;
- бінарний лог.

Текстове представлення даних зараз дуже розповсюджене через зручність формування та читання логу – файл можна відкрити за допомогою великої кількості текстових редакторів, кожна подія відображається окремим рядком.

Розширення .xml використовується коли система сконфігурована таким чином, що кожна подія представляє собою окремий набір вкладених записів, і структура логу нагадує дерево. Такі логи складніше інтерпретувати людині, для цього часто використовується додаткове програмне забезпечення.

Файли бінарного логу та формату CSV вже не розраховані для інтерпретації людиною, але такий формат дуже зручний для обробки додатковим програмним забезпеченням.

Також необхідно визначити й задати єдиний формат даних, достягти консистентності представлення різнотипової інформації. Це складне питання, яке вирішується на етапі проектування системи, оскільки зміни у вже запровадженому форматі даних потягнуть за собою переконфігурацію системи, що є тривалою та догорою операцією.

Основні принципи, якими керуються при визначенні формату – результуючий запис має бути максимально лаконічним та повністю описувати подію. Складнощі викликає визначення цілісності опису події, але існують розповсюджені практики, підходящі для більшості систем. Наприклад, запис починається з шляху де розміщений файл у файловій системі, вказання в записі точного часу до сотих (часто - до тисячних) за Всесвітньо координованим часом, тощо.

Насамкінець залишається питання зберігання логів, а точніше, їх заміни. Оскільки нема ні можливості ні потреби зберігати всі логи вічно, створюють правила видалення старих та заміщення їх новими. Зараз виділяють чотири найпопулярніших підходи до правил заміщення:

- заміщення по часу;
- заміщення по розміру файлу;
- заміщення, пов’язане з перезавантаженням серверу;
- динамічне заміщення, пов’язане з кількістю звернень до файлу.

Заміщенню по часу віддають перевагу найбільше. Підхід простий – встановлюється певний проміжок часу(найчастіше це декілька діб), довше якого дані не зберігаються та можуть бути перезаписаними новими. Проміжок

часу обирається в результаті аналізу наскільки довго отримані дані не будуть втрачати своєї актуальності та нестимуть будь-яку цінність.

Заміщення по розміру файлу – більш гнучкий підхід. Його перевага полягає в тому, що розмір файлу гарантовано не вийде за задані общемеження, обсяг зайнятої ним пам’яті завжди очікуваний. Проте вагомим недоліком виступає неочікуваність форматування файлу даних. Звісно, момент видалення можна приблизно розрахувати, але залишається ймовірність невчасної втрати важливих даних.

Заміщення по перезавантаженню сервера - досить рідко використовуване правило, тому що більшість систем, що використовують процес логування не потребують регулярного перезавантаження серверів. Це призводить до того, що файлів логів стає багато, їх інформацію вже недоцільно використовувати і при цьому вони займають велику кількість пам’яті, що, як мінімум, сповільнює сервіс.

Динамічне заміщення, орієнтоване на кількість звернень до файлу вважається найскладнішим правилом. Його проектування полягає в визначенні критичної кількості надходження нових даних (звернень до файлу), після якої раніше збережена інформація вже втратить актуальність. Впровадження такого правила потребує серйозної аналітики, та при оптимально заданих параметрах, воно дійсно забезпечує зменшення розміру файлу та вчасне його оновлення. Проте, через складність впровадження та несуттєвий виграш в пам’яті, частіше надають перевагу заміщенню по часу.

В сфері виробництва, а саме в технологічних процесах, логування зазвичай реалізоване через зв’зок сервера з окремою машиною, що слугує монітором. Вона збирає дані технологічного процесу, приводить до одного формату та відправляє на сервер. На теперішній час немає єдиного способу збору даних, оскільки технологічне оснащення принципово відрізняється від одного підприємста до іншого, а окремі автоматизовані та автоматичні прилади вже в своїй структурі можуть мати системи легування або інтерфейси для зв’яку з сервером.

1.2 Аналіз існуючих рішень

На сьогоднішній день на ринку представлено багато систем, що мають деякий функціонал, необхідний для аналізу та відображення внутрішньої інформації технологічного процесу. Їх спільна недосконаліть, як рішення поставленої задачі, полягає в тому, що вони використовуються для вузькозаданих цілей, і, як наслідок, реалізація їх зручного та одночасного використання тягне за собою складний процес інтеграції не тільки з технологічним процесом, а і між собою.

1.2.1 Системи диспетчерського управління та збору даних

Найбільш популярні рішення поставленої задачі пропонуть системи, виконані на основі SCADA. Аббревіатура SCADA розшифровується як «диспетчерський контроль та збір даних» (в перекладі – Supervisory Control And Data Acquisition) і являє собою набір програмних інструментів який використовують для розробки систем моніторингу, обробки та збереження даних про об’єкт керування або технологічний процес. Такі системи найчастіше запроваджують на виробництвах, які потребують постійного контролю технологічного процесу. Для збору даних системи використовують драйвери введення/вивденення та OPC протоколи. Часто виробники також пропонують додаткове програмне забезпечення для полешення конфігурації конроллерів, лічильників, та інших елементів, з яких зчитується інформація.

Пакет SCADA допомагає вирішити наступні задачі:

- збір та збереження даних технологічного процесу на віддаленій машині в режимі реального часу;
- обробка та відображення інформації за допомогою користувацького інтерфейсу;
- полегшення управління технологічним процесом та формування звітності в сконфігурованому вигляді.

Не зважаючи на те, що пакет SCADA має багато можливостей та за допомогою нього можна сконфігурувати систему для практично будь-якого виробництва, найчастіше прибігають до закупівлі готових систем та їх інтеграції з технологічним процесом.

З точки зору проектування системи, пакет SCADA має наступні переваги:

- різноматний набір інструментів для розробки надає можливість створити систему практично для будь-якого технологічного процесу або об'єкту керування;
- велика кількість зовнішніх інтерфейсів введення/виведення полегшують інтеграцію системи з технологічними об'єктами;
- програмний пакет надає гнучкі можливості вибору архітектури системи. В залежності від специфіки технологічного процесу, впроваджується автономна, клеєнт-серверна або зрозподілена архітектура.

Вагомим недоліком всіх SCADA-систем вистуває вразливість до хакерських атак сервісу SACADA WebServer до версії 2.03.0001. Про цю вразливість було завлено в 2016 році організацією ICS-CERT[2]. В ході атаки зловмисники проводили маніпуляції з браузером жертви та в результаті завантажували URL, створений стороннім додатком, який містив шкідливий скрипт, написаний мовою програмування JavaScript. Вразливість була визнана критичною, оскільки надавала контроль технологічним процесом. Після інциденту було випущене оновлення SCADA WebSrver, проте контролери зі вже встановленим шкідливим програмним забезпеченням відновленню не підлягають.Відносно останніх оновлень сервісів SCADA вразливості не були задокументовані.

Однією з найпопулярніших SCADA-систем на ринку України визнають Simple-Scada. Система з'явилась на ринку в 2014 році, зараз представлена її актуальна версія 2.3.5.0, вона має три доступні ліценції, які відрізняються кількістю зовнішних тегів (можливих для взаємодії елементів).

Інтерфейс однієї з реалізованих за допомогою неї систем зображено на Рисунку 1.

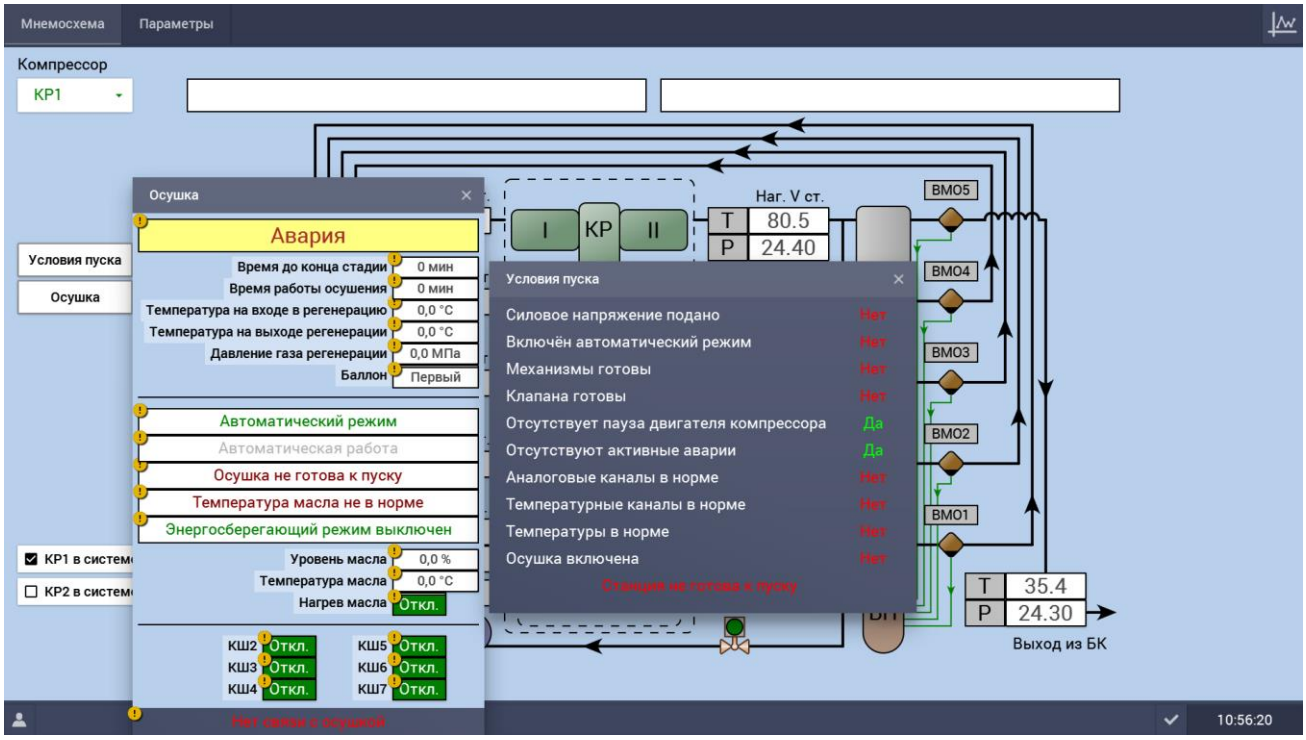

Рисунок 1 – Користувацький інтерфейс системи, розробленої для АСУ ТП Абсорбаційного осушення газу[3].

Користувацький інтерфейс може суттєво відрізнятись між системами, оскільки Simple-Scada надає додаткове програмне забезпечення для налаштування графічних мнемо-схем, побудови та відображення графіків, анімацій, тощо. Основна логіка системи конфігурується в трьох программах : для клієнтської сторони, серверної та для редагування проекту. Налаштовувати систему можна як за допомогою детального графічного інтерфейсу так і шляхом написання автоматичних скриптів мовами програмування Pascal та Delphi.

В ході аналізу характеристик системи Simple Scada буди виділені наступні переваги:

- низькі необхідні системні вимоги;
- зручний користувацький інтерфейс;
- організація архітектури з необмеженою кількістю клієнтів;
- автоматичне відображення на моніторі, формування звітів та подача звукових сигналів при виході технологічних параметрів за контрольовані границі;

– можливість налаштування прав користувачів;
– можливість логування сповіщень на сервері;
– наявність детальної документації.

Система також має функції збору та відображення статистики зміни значень параметрів – «трендів», її зберігання в базах даних MySQL в режимі реального часу.

Були виявлені наступні недоліки:

– система добре підходить для управління об’єктом керування, але для технологічного процесу труднощі викликає навіть конфігурація мнемо-схем;
– у випадку, коли нетипова задача не має рішення наданого розробниками системи, її розв’язок потребує великої кількості трудовитрат;
– значне зниження продуктивності системи, якщо проект має багато елементів або великі обсяги даних оброблюваних даних;
– вихідний код програмного забезпечення закритий для публічного доступу;

Враховуючи сукупність наведених факторів, систему Simple-Scada можна оцінити як зручну, гнучку та достатньо функціональну систему для простих технологічних процесів та типових об’єктів керування.

Для складних технологічних процесів вона буде ергономічно невигідною через високу вартість(як матеріально так і за витратами часу та праці) інтеграції та непродуктивність роботи зі складними проектами. Також, інциденти пов’язані зі знайденими вразливостями сервісів SCADA та закритий вихідний код програмного забезпечення позбавляють впевненості в інформаційній захищеності даної системи.

1.2.2 Система відображення даних Grafana

Grafana являє собою потужний інструмент моніторингу та візуалізації даних технологічного процесу. Найбільшу популярність сервіс здобув в сфері інформаційних технологій, проте він не має обмежень або специфічних

налаштувань до використання тільки в даній області.

Основні два поняття, якими оперує Grafana – це дошка та панель. Панель виступає в ролі опорного елементу відображення. На панелі розміщуються графіки, таблиці, матриці та інше. Дошка представляє сукупність панелей, що об'єднані спільними параметрами, наприклад, назва бази даних або таблиці в ній, датчик, з якого зчитується інформація, тощо. Інструмент надає гнучкі налаштування панелей та дошок.

Основною конкурентною перевагою Grafana виступає її інтуїтивно зрозумілий графічний інтерфейс та багатий вибір налаштувань відображення протікання процесів на виробництві. Користувацький інтерфейс сервісу зображений на Рисунку 2.

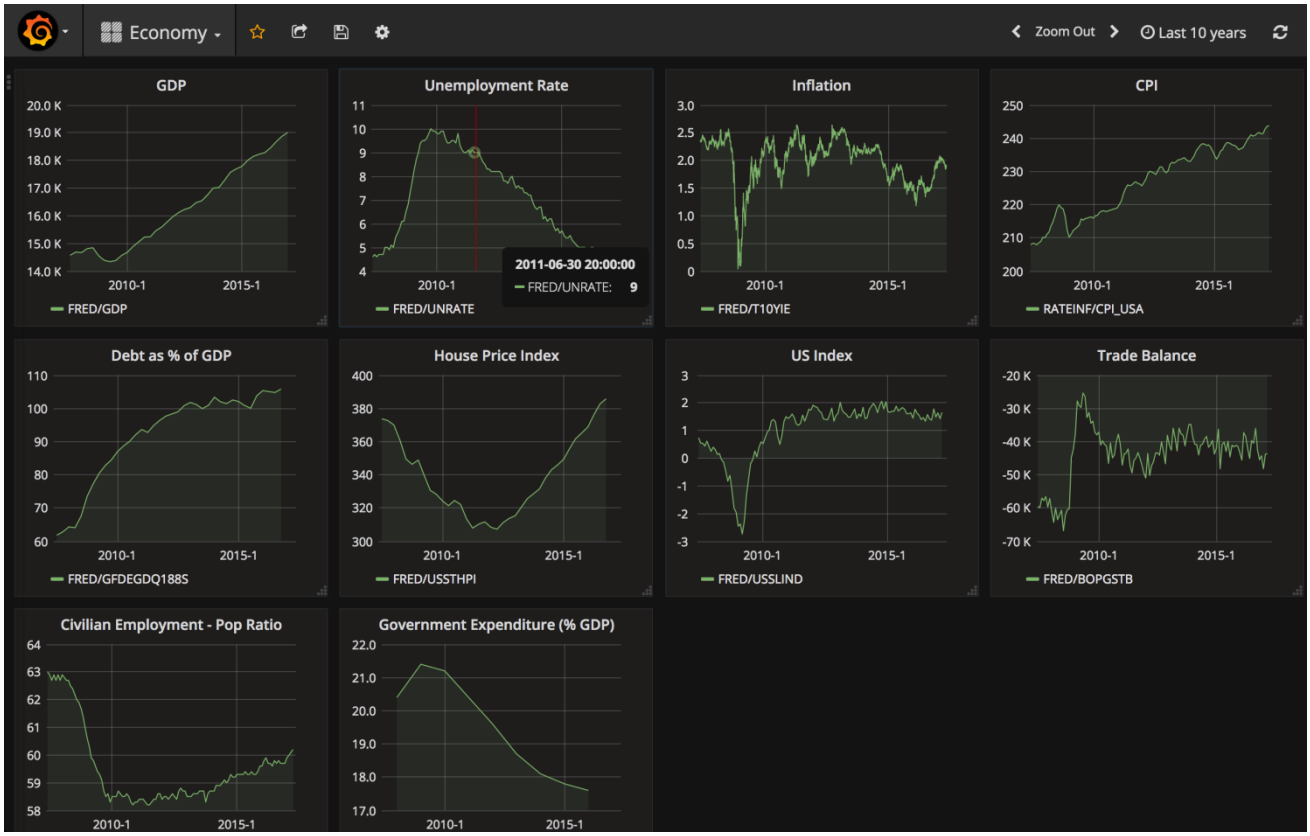

Рисунок 2 – Дошка створена за допомогою Finance Plugin[4]

З архітектурної точки зору розглядають дві можливі конфігурації. В першому випадку сервіс може виступати посередником між серверною та клієнтською сторонами, тобто він оброблює та виконує запити до бази даних,

відправлючи результати в браузер. У випадку, коли користувач вирішив обмежити права сервісу на доступ до бази даних, запит формується та відправляється безпосередньо з клієнтської сторони, а Grafana тільки оброблює результати.

Переваги інструменту:

- для налаштування графічного інтерфейсу панелі не потрібні знання мов програмування чи досвід в адмініструванні систем;
- готова система зручна у використанні, має багато мінімальних налаштувань за замовчуванням(наприклад сортування таблиць, розтягування графіків, тощо);
- має додаткове програмне забезпечення для інтеграції з зовнішніми додатками;
- можливість налаштування схематичних звітів;
- програмний код знаходиться в вільному доступі.

Також Grafana має функціонал тригерів – анотацій. Він використовується, коли необхідно відстежити поведінку системи (значення відповідних параметрів, відображення на екрані встановленого повідомлення, тощо) на певні події.

Недоліки інструменту:

- складність налаштування необхідних метрик, неочевидність інтерфейсу на етапі проектування;
- велика частина функціоналу надається додатковим програмним забезпеченням;
- завантаженість системи спричиняє затримки в відображенні даних;
- при специфічних налаштуваннях дошки система сповіщень може працювати некоректно.

Також, не зважаючи на багаті можливості в конфігурації відображення обраних метрик, вагомим недоліком Grafana виступають неповні можливості налаштування тригерів та системи сповіщення про їх спрацьовування.

Одна з особливостей системи сповіщення полягає в тому, що при спрацюванні тригеру на подію генерується два повідомлення – про його початок

та кінець. В налаштуваннях відсутня можливість редагування цієї поведінки, тому в частивні випадків така особливість може виступати недоліком, оскільки надає в два рази більше сповіщень, ніж необхідно, зашумлюючи робочий простір панелі.

Загалом, Grafana представляє зручний та естетичний інтерфейс для відображення та моніторингу даних. Інструмент інтегрований з розповсюдженими базами даних, підтримує більшість операційних систем, простий у встановленні.

Але, в контексті поставленої задачі, його використання надає лише частину рішення, оскільки функціонал аналізу невеликий та орієнтований на бізнес-аналітику.

Також велика частина функціоналу на малих та середніх підприємствах не необхідна та рідко використовується, це робить впровадження системи невигідним з економічної точки зору.

Висновки до розділу 1

В даному розділі була проаналізована предметна область технологічного процесу виготовлення шоколалу та процесу формування та збереження інформації технологічного процесу.

Також були розглянуті два з найпоширеніших наявних рішеннь проблеми відображення та аналізу даних виробництва – системи Simple Scada та Grafana. Було проаналізовано їх переваги та недоліки, на основі яких були визначені вимоги до розроблюваної системи.

Перш за все, одна з основних поставлених вимог до системи – це відображення даних технологічного процесу в зручному для людини форматі. Ця функціональна можливість є фундаментальною перевагою розглянутих вище рішень, в порівнянні з іншими, представленими на ринку. Спільним недоліком вказаних раніше систем виступає незручність користувацького інтерфейсу на етапі налаштувань відображення даних (Grafana) або на момент їх відображення

(Simple Scada) – надмірна детальність, враховуючи специфіку виробництва, може суттєво ускладнювати конфігурацію системи або створювати відображення інформації непридатним для обробки людиною.

Оскільки розроблювана система розрахована, в першу чергу, на використання працівниками, які безпосередньо взаємодіють з тенологічним процесом, вимогою до системи було поставлено забезпечення зрозумілого на інтуїтивному рівні та лаконічного користувацького інтерфейсу, який не потребує спеціальних вмінь для роботи з ним.

Також, розглянуті системи мали ряд недоліків, наприклад складність в інтеграції з параметрами технологічних операцій, помилкові спрацювання систем сповіщення, тощо.

Велика їх частина пояснюється тим фактом, що системи були створені без орієнтації на потреби конкретного виробництва та з урахуванням можливості їх інтеграції в практично будь-який технологічний процес. Це призводить до високої складності їх впровадження та неповної відповідності їх функціональних можливостей потребам виробництва.

Такі недоліки створюють попит на вузькоспеціалізовану систему для конкретного виробництва з можливістю її розширення. У випадку технологічного процесу виробництва темного шоколаду, можливість розширення системи дуже важлива, оскільки вона дозволяє ефективно використовувати дану систему при внесенні змін до основного рецепту шоколаду, або розчиренні чи модифікації технологічного процесу.

2 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ

Проект був розроблений за допомогою використання мови Java та додатково підключеного наступного програмного забезпечення: JFreeChart та Maven, в якості середи розробки було використано Eclipse IDE.

2.1 Мова програмування Java

На данний момент існує багато мов програмування і з кожним роком створюються нові. Таке різноманіття пояснюється тим, що, формально, мова програмування – це знакова система, що надає свої можливості та накладає свої обмеження в роботі з машинним кодом. В розрізі такої парадигми стає очевидно, що винайти ідеальну мову програмування для кожної поставленої задачі неможливо. Саме тому, для стрімкого розвитку технологій нехобхідний не менш стрімкий розвиток інструментів, якщо існуючі роблять рішення неоптимальним та не ефективним в процесі розробки.

За результатами нещодавньої оцінки, станом на травень 2019 року, Java посіла перше місце в рейтингу найбільш популярних мов програмування.[5] Вона також поділяє перше місце з JavaScript у рейтингу найбільш часто використовуваних для розробки мов в Україні.[6] Світовій визнаності передує довга історія розробки та модифікації мови.

2.1.1 Основні відомості про мову Java

На етапі створення мови програмування Java, Джеймс Гослінг планував створити зручний інструмент для програмування побутових приладів. Спершу передбачалось, що платформа, як і програмне забезпечення, повинні бути невеликих розмірів через обмеження пам’яті та енергії, що можуть надати побутові прилади. Також амбіційні плани розробників включали платформо-незалежність скриптів, тобто, незалежність від архітектури процесора пристрою.

Розробка Java почалась в 1991 році компанією Sun Microsystems, а перша версія – Java Development Kit 1.0 була випущена в 1996 році. За цей час було розроблено основу архітектури мови. Також було знайдене рішення як зробити програмний код платформо-незалежним. Це було досягнуто введенням додаткового рівня обробки програмного коду – віртуального.

Ідея полягає в тому, що, перед виконанням, вихідний код програми транслюється компілятором в байт-код, який інтерпретується та виконується за допомогою віртуальної машини - Java Virtual Machine (JVM). Оскільки байт-код – це псевдо-код і він не орієнтований на конкретну архітектуру процесора, це зробило можливим виконання коду на будь-якій фізичній машині, при умові встановленої JVM. Такий підхід породив лозунг “Write once, run anywhere”.

Пізніше була створена специфікація процесорів, машинний код яких – двійковий код. Хоча час виконання програм, написаних мовою Java, на них був до двадцяти разів менше, ніж на інших процесорах, вони не здобули популярност.

Не дивлячись на те, що перша версія мови вирішувала багато фундаментальних питань, наприклад управління пам’яттю, процес перетворення коду в байт-код, розробка першого компілятору, та багато іншого, розробка програмного забезпечення (не в навчальних цілях) з використанням Java 1.0 була практично неможливою – функціональних можливостей було замало для цього.

Тому через рік з'явилась наступна стабільна версія (JDK 1.1). Вона багато в чому доповнювала першу версію – це впровадження стандарту з'єднання з базами даних, підтримка Unicode, підтримка аудіо-файлів, тощо. Також, було впроваджено ще одне масштабне архітектурне рішення, яке після багатьох оптимізацій використовується і в останніх версіях. Мова йде про динамічний Just In Time компілятор (JIT-компілятор).

Як було зазначено вище, програмний код, при запуску програми, проходить декілька операцій перетворення та інтерпретації. Перед виконанням компілятор перетворює його в байт-код. Віртуальна машина вже безпосередньо виконує інструкції коду.

В такій системі JIT-компілятор додатково перетворює деякі сегменти отриманого двійкового коду в машинний код під час роботи програми. Такий підхід був впроваджений для трансляції незмінних об’єктів, методів та классів, на які багато посилань. Обмеженню підлягали сутності, які модифікувались в процесі виконання коду.

Оскільки машинний код виконується процесором значно продуктивніше, ніж інтерпретуємий код, використання JIT-компілятору суттєво зменшило час виконання програми та підвищило ефективність використання пам’яті (певною мірою, за рахунок збереження машинного коду – створення кешу даних, який міг надалі використовуватись при перекомпіляції коду).

Наступні декілька версій в більшій мірі були направлені на оптимізацію внутрішніх алгоритмів платформи. В цьому була наявна потреба, оскільки програмний код, написаний з використанням перших версій Java виконувався до дев'яти разів довше ніж ті самі інструкції написані мовою C.

Починаючи з п'ятої версії, синтаксис мови почав стрімко розширюватись новими конструкціями. Офіційна специфікація Java 5 з'явилась в 2004 році. Цікаво те, що мова Java не є самобутньою. Хоча деякі конструкції принципово відрізняються від інших платформ, частина функціоналу інших мов була визнана успішною та реалізована в Java по аналогії з ними.

Так, наприклад, в п’ятій версії, реалізовано подібно до реалізації в мові C++, був впроваджений спеціальний тип – перелічувальний (в перекладі – enum). Одночасно з ним з'явились інструменти узагальненого програмування (generic), що суттєво відрізняються від стандартів C++.

В 2014 році відбувся реліз Java SE 8. Ця версія набула найбільшого визнання. Довгий час розробки мови заккумулювали в восьмій версії оптимально розроблені алгоритми компіляції, виконання коду та багаті функціональні можливості.

Наступна версія – Java SE 11 з’вилась в 2018 році, на сьогоднішній день вона остання зі стабільних. Велика частина її нововведень – оновлення архітектурних аспектів, такі як експериментальне впровадження нової системи

збирача сміття (в перекладі - garbage collector), стандарт роботи з HTTP та HTTP2, підтримка Unicode 10, тощо. Також були розширені функціональні можливості мови – оновлення системи модифікаторів доступу, поява нового зареєстрованого слова – var та впровадження його використання в лямбда-функціях, та інше.

Проте зараз все ще велика частина комерційних організацій використовує восьму версію Java для розробки. Це пов'язано з рядом причин, одна з головних – це непередбачливість оновлення. Хоча, один з основних критеріїв стабільності оновлень – це підтримка звортньої сумісності версій тобто, можливість використовуючи нову версію мови безперешкодно звертатись до механізмів попередньої, він не завжди виконується повністю. Тому, особливо на великих проектах, практично неможливо оцінити успішність оновлення до нової версії та необхідний для цього час.

В порівнянні з іншими популярними мовами програмування, нові функціональні можливості в мові Java впроваджуються дуже вивірено та поступово. З одного боку, такий консервативний підхід не завжди зручний для розробників, оскільки для деяких вони мають повторно «винаходити» рішення, які вже можуть бути впроваджені в інших мовах програмування механізмами платформи. З іншого боку, він надає впевненість у надійності та оптиміованості існуючого функціоналу та зменшує ризики при оновлені до нової версії.

2.1.2 Компоненти платформи Java

Перший базовий компонент Java – це компілятор. Компілятор являє собою програмне забезпечення, основні функції якого – це трансляція вхідного коду, написаного високорівневими мовами програмування, у псевдо-код. Також, трансляція може відбуватись одразу до машинного коду. Компіляція Java-коду - трохи складніший процес, його обуло оисано вище, в розділі 2.1.1. Деякі архтектурні особливості платформи були запозичені. Наприклад на момент проектування компілятору Java, рішення проблеми кросс-платформеності вже

інсувало в реалізаціях мови Pascal і ідея була запозичена.

Також важлива фукція компілятора – це збирання проекту. Зібрання (в перекладі - assembly) – компонування вже трансльованого коду у виконачий файл. В кінцевому вигляді файл містить текст програми, додатково підключені бібліотеки, які необхідні для виконання коду та додаткову мета-інформацію (наприклад, врсію програми). Хоча платформа вже має такий функціонал, часто розробники використовують стороннє програмне забезпечення для збирання проекту. Таке програмне забезпечення надає можливості детальної конфігурації процесу збирання проекту, наприклад, визначення послідовності кроків збирання (очищення системних файлів, збережених з останнього збирання, запуск модульних тестів, тощо) та їх налаштування. Така перевага особливо суттєва при розробці на підприємстві, де якість коду підляє детальному контролю, оскільки допомагає автоматично перевірити велику частину механічних помилок розробника.

Необхідність інтерпретації трансльованого коду суттєво знижує якісні показники виконання програми – збільшує час вионання, підвищує обсяг займаємої пам’яті, тощо. Тому, з метою оптимізації, платформа Java також використовує метод динамічної компіляції, а для цього – додаткове програмне забезпечення - JIT-компілятор.

Інший важливий компонент платформи – віртуальна машина Java. JVM являє собою фундаментальний модуль виконачої середи (Java Runtime Environment, або скорочено - JRE). Основних функцій модуля дві:

- інтерпретація байт-коду та його виконання;
- виконує управління пам’яттю, яку займає проект та оптимізує її.

Управління пам’яттю програми – складний та специфічний процес. Пам’ять, яку використовує програма, для оптимізації, розділяють на дві області – стек(в перекладі - stack) та купа(в перекладі - heap).

Модель стеку організована за принципом LIFO. Це означає, що пам’ять для нових об’єктів буде виділена на початку (або «вершині») стеку. Стек використовується для зберігання посилань на об’єкти та змінні, а самі об’єкти

зберігаються в купі. Коли потік виконання програм виходить з «області видисоті» об’єкту, посилання на нього видаляється. Така організація пояснюється тим, що розмір пам’яті стеку набагато менший пам’яті купи, а через оптимальну організацію, доступ до нього набагато швидший ніж до купи.

Область пам’яті купи організована складніше. Її можна умовно розбити на дві секції – область де розміщуються нещодавньо створені об’єкти та область для об’єктів, які необхідні на протязі довгого часу роботи програми. До впровадження Java 8 існувала третя секція – для зберігання додаткової інформації про класи та методи.

Ця пам’ять є спільною для всієї програми, для автоматичного очищення був впроваджений механізм збирача сміття (в перекладі – garbage collescor). Його ідея полягає в тому, що він аналізує кожен об’єкт, який знаходиться в купі на момент його проходу, на предмет того, чи необхідний бін буде для подальшої роботи програми (тобто, чи є на нього посилання у коді, який ще не виконувався). В результаті аналізу він може його видалити (якщо посилань на об’єкт більше нема) або залишити. Проблема алгоритму полягає в тому, що неможливо передбачити, коли буде викликаний збирач сміття та чи точно він видалить той чи інший об’єкт.

Над алгоритмами оптимізації стеку, купи та збирача сміття довго працювали та їх оновлення випускають практично в кожній новій версії Java. Це пов’язано з тим, що розробник безпосередньо не має доступу до пам’яті, не може ініціювати видалення об’єктів коли йому це потрібно, інакше кажучи, не може самостійно нею управляти, як при використанні більш низкорівневих мов, наприклад, мови програмування С.

При цьому, якщо було виділено замало пам’яті, або вона неоптимально використовувалась (наприклад, у випадку логічної помилки в коді), та обсяг вільної пам’яті закінчився в процесі виконання коду, то генерується повідомлення про помилку (для стеку та купи генеруються різні повідомлення, що спрощує процес пощуку помилки) та програма аварійно завершує свою роботу.

Загалом, платформа Java несе в собі сукупність декількох основних компонентів – JRE та JDK, які об'єднують множини окремих модулів. Не дивлячись на таке розділення, на практиці найчастіше встановлюють та використовують більш повний модуль – JDK.

Цілісні компоненти та модулі платформи досить детально відображає діграма з офіційної документації платформи Java зображена на Рисунку 3.

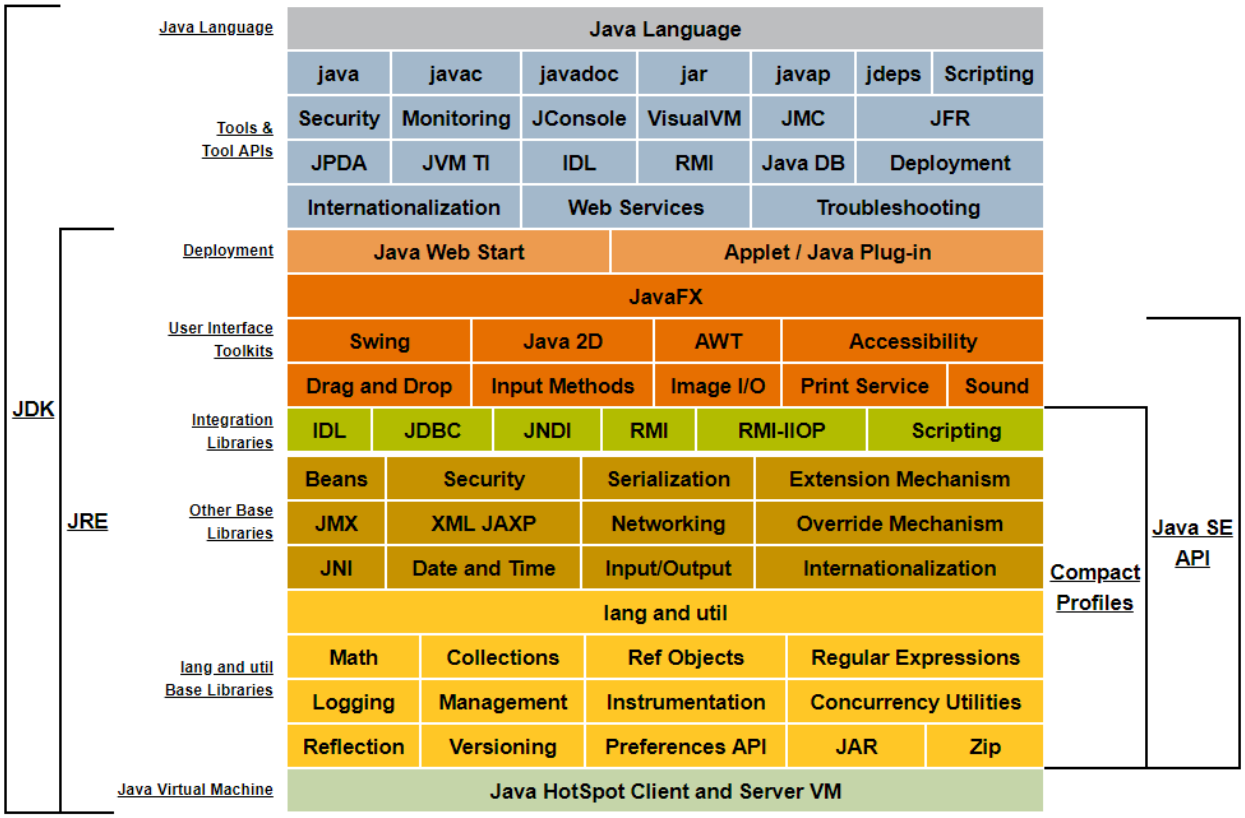

Рисунок 3 – Об'єднання технологій платформи в компоненти[7]

Як видно з діаграми, JRE, що згадувалось вище, складається не тільки з віртуальної машини, а також містить додаткові бібліотеки та класи. Його важливим компонентом висупає Java Plug-in, оскільки він надає функціонал роботи з браузерами.

JDK містить в собі все програмне забезпечення, яке входить до JRE, а також розширюється додатковими модулями такими як компілятор та інструмени налагодження – дебагер (в перекладі - debugger).

Загалом, пакет JDK володіє всіма необхідними технологіями для розробки

програм мовою Java(з можливістю підключення поключення комонентів, написаних з виористанням іншої мови програмування) та виконання програм за допомогою компоненту JRE.

Функціональні можливості мови програмування Java дозволяють сворювати широкий спектр спецефічних програм. В цьому допомагає об'єднання технологій в так звані платформи в залежності від специфіки їх використання. Зараз платформа Java підтримує чотири видання:

- Java SE;
- Java EE;
- Java ME;
- Java Card.

Java Standart Edition – ключове видання. Спроектоване для індивідуальної розробки, або для розробки на невеликих підприємствах. Включає в себе всі основні технології Java, такі як компілятор, JRE, підтримка основних класів та бібліотек, тощо.

Java Enterprise Edition являє собою розширену версію Java SE. Видання нещодавньо змінило назву на Jakarta, а його модулі, починаючи з одинадцятої версії Java, були видалені зі Java SE. Ця версія платформи представляє собою додатковий набір технологій, необхідний для розробки програмного забезпечення на великих підприємствах. Велика його частина, орієнтована на веб-розробку, специфікацію серверних компонентів.

Java Micro Edition включає в себе набір технологій для розробки програмного забезпечення для пристроїв, які мають невеликі об'єми пам'яті та потужність процесору – наприклад, смартфонів. Розробка для таких приладів накладає багато обмежень, а дана версія платформи допомагає зробити їх менш критичними впроваджуючи налаштування для JVM та набір основних класів.

Java Card – найбільш специфічна версія платформи. Вона використовується для розробки програм, які будуть виконуватись на дуже слабких приладах, з точки зору потужності та пам'яті. Специфіка даної конфігурації полягає в тому, що і ній впроваджені зміни до алгоритму

формуванняя байт-коду.

Раніше існувала ще одна конфігурація платформи – Personal Java, але її зараз не пітримують, а функціональні можливості буди додані в конфігурацію Java ME.

Не двилячись на широкі функціональні можливості мови Java, для розробки програмного забезпечення на підприємстві цього недостатньо. Тому існує велика кількість фреймворків(в перекладі - framework), які використовуються в процесі розробки в залежності від специфіки поставленої задачі. Фреймоврк – це додатковий програмний пакет призначений для використання в процесі розробки, який надає більш специфічний функціонал, як доповнення до основної мови програмування.

Один з найбільш популярних фреймворків які використовуються разом з мовою Java – Spring Framework. Він призначений для полегшення розробки додатків з клієнт-серверную архітектурою. Spring являє собою дуже масштабне рішення з модульною структурою, це надає можливість за потреби підключати та використовувати тільки откремі модулі.

Також існує багато інших фреймвоків, призначених для розробки веб-додатків, наприклад – Blade, Google Web Toolkit, JHipster, тощо.

Також мова програмування Java використовується не тільки для розробки клієнт-серверних додатків. Вона вважається класичною й в сфері автоматизованого тестування додатків. Ця область зараз активно розвивається. Для організації автоматизованого тестування існують свої специфічні підходи та практики, окремі фреймворки, та шаблони проектування.

Зараз одна з найпопулярніших технологій в сфері тестування з використанням мови Java - Selenium Framework та група технологій, що побудовані на його основі(наприклад, Selenide).

Синтаксис мови Java був запозичений переважно з мов C та C++, при зрозумілих назвах створних розробником класів та змінних програмний код дуже легко читаємий.

2.2 Програмна бібліотека JFreeChart

JFreeChart представляє собою програмне забезпечення з відкритим вихідним кодом яке надає багатий функціонал для роботи з графіками.

Функціональні можливості включають наступні:

- налаштування побудови та відображення всіх основних видів графіків(лінійні, кругові, гістограми, графіки Ганта та інші);
- побудова та відображення декількох графіків на одній панелі;
- налаштування відображення графіків;
- налаштування інтерактивності графіків.

Загальний процес побудови графіку досить простий. Він складається з наступних послідовних кроків:

- збір даних та їх валідація, формування об’єкту DataSet;
- визначення типу графіку та налаштування його характеристик;
- створення графіку;
- представлення графіку на екрані та/або формування файлу з роширенням .png або .jpeg.

Серед подібних бібліотек JFreeChart вирізняють багато переваг. По-перше, її можливості в налаштуваннях відображень графіків дуже детальні, конфігуруються практично всі параметри. По-друге, вона знаходиться у вільному доступі (тобто, безкоштовна) та має відкритий вихідний код[8].

Недоліки JFreeChart наступні:

- з великими об’ємами даних було помічено зниження швидкості побудови графіків;
- з оновленням до нових версій виникають труднощі, обернена сумісність підтримується не повністю, існують навіть окремі гайди та покрокові рекомендації на цей випадок.
- довгий час документація до використання була платною. Нещодавно з’явилась у вільному доступі офіційна скорочена версія документації, але вона недостатньо інформативна.

2.3 ПЗ для збирання проекту Apache Maven

Apache Maven являє собою програмне забезпечення, яке призначене для автоматизації процесу збирання проекту, написаного мовою програмування Java. Конфігурація процесу збирання описується розробником в спеціальному файлі, з розширенням .xml, за допомогою мови Project Object Model(або, скорочено - POM). Структура POM-файлу детально та строго специфікована. Обов'язкові налаштування, які мають бути описані це:

- номер весії зібрання;
- номер версії конфігурації;
- назва групи, до якої належить даний проект;
- назва проекту.

Використовуючи Apache Maven, можна створювати особливу систему конфігурації проектів, об'єднуючи їх в групи, та задаючи спільні налаштування. Також існує багато необов'язкових налаштувань – це підключення сторонніх бібліотек, управління процесом збирання, тощо. Збирання проекту за допомогою Apache Maven проходить шість кроків[9]:

- компіляція проекту;
- запуск тестів;
- пакування проекту (формування .jar/.war/.ear файлу);
- запуск інтеграційних тестів;
- інсталяція;
- публікація зпакованого проекту на віддаленому сховищі.

Кожен з даних кроків може мати додаткові налаштування, а в разі виникнення помилки на будь-якому з них, процес збирання завершується з описом помилки.

Однією з суттєвих переваг Apache Maven висупає можливість ініціювання збирання проекту з терміналу командного рядка. Ця можливість необхідна, в разі розміщення проекту на сервері. Також вона дозволяє ініціювати неповний процес, а тільки його окремий крок. В цьому випадку будуть пройдені всі кроки,

що йому передують.

Іншою вагомою перевагою Maven є можливість управління залежностями. Такий функціонал спрощує підключення додаткового програмного забезпечення в проект. Його алгоритм полягає в тому, що розробнику достатньо описати в pom–файлі залежність, вказавши ім'я резпозиторію, де знаходиться необхідне програмне забезпечення та номер його версії(більшість популярних бібліотек мають такі репозиторії). Подальшу роботу використовує Maven – відбувається перевірка існування шуканого програмного забезпечення в локальному репозитрії(він створюється при інсталяції Maven). Якщо воно не було завантажене раніше, завантажується з віддаленого репозиторію. Управління теж включає автоматичне вирішення конфліктів версій, видалення залежності в разі логічних помилок її підключення, тощо

Також особливістю Apache Maven виступає декларативність збирання проекту. Це означає, що в pom-файлі конфігурації знаходиться тільки налаштування побудови та збирання проекту, безпосередніх команд, що виконуються на будь-якому кроці збирання не допускається.

На додачу, Maven надає можливість швидкого створення нового проекту з вже заздалегідь заданими конфігураціями. Така можливість надається архітипами, тобто, шаблонами, каркасами проектів. Архітип задає стандартну структуру розміщення файлів проекту та заздалегідь підключені залежності. Їх кількість не уклінно зростає, на момент написання проекту існує 2423 доступні архітипи. З їх використанням новий проект створюється за допомою комади « mvn archetype : generate » та вказанням базових конфігурацій – версій та назви проекту.

Проект, який використовує зібрання за допомогою Apache Maven може відкриватись в будь-якій середі розробки. Також Maven не має залежності від операційної системи. Ці фактори роблять процес обробки проекту легкими в переміщенні та, в деякій мірі, стандартизованими.

Такі переваги та особливості роблять фреймворк Apache Maven зручним інструментом розробки та підтримки програмного забезпечення.

2.4 Середа розробки Eclipse IDE

Eclipse IDE представляє собою середу розробки програмного забезпечення.[10] Довгий час вона була найбільш популярна серед Java розробників. Таке визнання базувалось на багатому функціоналі та зручному графічному інтерфейсі.

До переваг Eclipse IDE відносять:

- крос-платформеність, може використовуватись на всіх популярних операційних системах;
- велика кількість додаткового програмного забезпечення(plug-in), які допомагають не тільки оптимізувати процес розробки, а й інтегрувати Eclipse IDE з іншими додатками та системами;
- багато фунцій, які допомагають в процесі розробки та налагодження коду(наприклад, можливість автоматичної генерації стандартних шаблонів коду);
- детальна офіційна документація та велика кількість навчальних матеріалів;
- відкритий вихідний код та безкоштовна ліцензія.

Також існує ряд особливостей Eclipse IDE які принципово відрізняють її від аналогічних рішень, та можуть виступати як перевагами так і недоліками.

Одна з таких особливостей - це нестандартність конфігурації компілятора. Вона полягає в тому, що він часто викликається ще в процесі написання коду (наприклад, завантаження файлу ініціює виклик компілятору). Така система досить зручна, оскільки дозволяє відразу помітити синтаксичні помилки в коді. Але її недоліком виступає зниження продуктивності та швидкодії середи розробки.

По-друге, система плагінів Eclipse IDE дуже велика. Зараз вона нараховує близько 1700 плагінів. З одного боку таке різноманіття надає можливість сконфігурувати проект максимально підходящими інструментами. З іншого боку можлива ситуація виникнення конфліктів через несумісність декількох плагінів між собою. Тому, розробники зазвичай дуже помірковано встановлюють тільки

необхідні плагіни, за можливості уникаючи їх накопичення.

Також, часто розробники програмного забезпечення самостійно створюють плагіни для його підключення, та вказують їх в своїй документації як офіційні.

Висновки до розділу 2

В даному розділі були детально описані технології що використовувались для розробки системи відображення та аналізу даних технологічного процесу виготовлення шоколаду. В результаті детального аналізу предметної області були визначені вимоги до системи, задовільнити яким допомагають широкий спектр функціональних можливостей обраних технологій.

Окрім того, що обрані технології надають підходящі та зручні інструменти для розробки системи що відповідає визначеним вимогам, вони також мають багато функціональних можливостей, які допомагають сконфігурувати не тільки визначений необхідний функціонал системи, а й додатковий, який допоможе зробити систему більш зручною та ефективною з точки зору кінцевого користувача.

Наприклад, програмний пакет JFreeChart, який використовувався для побудови відображення вхідних даних системи, із великим об’ємом інструментів для дизайну та налаштувань графіків включає в себе можливість налаштувань інтерактивності графіків, можливість впровадження часткової їх конфігурації користувачем.

Також, Apache Maven, окрім своїх основних функцій – надання можливості конфугування процесу збирання проекту, робить можливим та зручним запуск системи із командного рядка. Враховуючи специфіку системи, така можливість не є необхідністю, проте вона надає додаткову варіативність в подальшому розширені системи.

3 АРХІТЕКТУРА РОЗРОБЛЕНОЇ СИСТЕМИ

Єтап проектування та побудови архітектури має фундаментальне значеня в процесі розробки будь-якої системи. На стадії перших кроків суспілства у використанні інформаційних технологій, появі мов програмування, більшість з яких зараз вважається застарілими, перед першими системами програмного забезпечення поставали зовсім інші питання архітектурної побудови. Переважно, вони полягали у вдалому виборі структури даних, виборі оптимального алгоритму з точки зору продуктивності та урахуванням архітектури процесору, тощо. З плином часу, стрімким розвитком інформаційних та комп'ютерних технологій та збільшенням області їх взаємодії з кожною людиою, системи ставали все масшабнішими, процес їх розробки та забезпечення взаємодії між іншими системами ставав все більш заплутаним та тернистим.

Єдиного визначення архітектури програмного забезпечення зараз не існує, деякі джерела наводять до 160 варіантів визначення. Однак, найбільш популярним вважаться визначення що, зазначене в стандарті IEEE 1417-2000[11]. В перекладі, архітектура програмного забезпечення представляє собою фундамент організації системи, який складається з компонентів, їх взаємодії між собою та зовнішньою середою, та підходів, які визначають її дизайн та розширення.

Вдало побудована архітектура програмного забезпечення має задовільняти наступним вимогам:

- працездатність системи, ефективність виконання поставлених задач в різних умовах;
- продуктивна одночасна розробка програмного забезпечення декількома розробниками;
- можливість доповнення системи новими функціональними можливостями при мінімальних витратах зусиль;
- можливість зміни або доповнення існуючого функціоналу при

мінімальних змінах в системі.

Відповідність архітектури програмного забезпечення таким вимогам робить процес розробки максимально ефективним та гнучким.

Основна проблема, яка постає перед розробником архітектури полягає в тому, що немає єдиного правильного рішення, методу або підходу, який буде максимально підходящим під кожну конкретну поставлену задачу. Існує велика кількість матеріалів та рекомендацій в цій сфері, проте більшість корисних вмінь здобувається з досвідом розробки програмного забезпечення.

Зараз одним із найбільш розповсюджених шаблонів проектування систем, побудованих з використанням об’єктно-орієнтованих мов програмування, виступає SOLID, запропонований Робертом Мартіном. Аббревіатура складається з назв п'яти принципів – Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion.

Принцип єдиної відповідальності заключається у визначенні для кожного об’єкту тільки однієї мети, для якої він використовується. Вся функціональність об’єкту повинна бути направлена тільки на виконання поставленої мети та інкапсульована в класі. Невідповідність такому принципу часто приводить до того, що сутності стають занадто великими та громіздкими, їх підтримка та модифікація стає неможливою без переписування вже існуючого функціоналу.

Принцип відкритості-закритості пропонує підхід використання програмних сутностей, за якого вони повинні мати можливість доповнення новим функціоналом, але при цьому розширення їх поведінки не повинно впливати на код, який з ними не пов’язаний. Пропонується реалізація принципу через використання механізму наслідування – основна сутність не модифікується , зміни вносяться в клас, що її наслідує. Такий підхід особливо зручний у використанні на підприємстві, де часто впроваджена особлива політика внесення нового коду в програмне забезпечення(наприклад, обов’язкова вимога завірення коду членом команди – code review, дописання юніт-тестів, тощо), оскільки може суттєво скоротити час на внесення змін.

Принцип підстановки Лісков полягає в проектуванні системи типів таким

чином, щоб функціонал типів-нащадків не конфліктував з функціоналом базового типу. Правило підстановки, при такому підході, полягає в тому, що об’єкт базового типу повинен мати можливість бути заміщеним об’єктом підтипу без втрати первісної функціональності та змін в програмному коді. В разі невідповідності цьому принципу, система втрачає суттєву частку гнучкості та можливостей розширення, втрати повного розуміння за яку поведінку відповідає сутність.

Ідея принципу розділення інтерфейсів пропонує уникати ситуацій, коли інтерфейси призначенні для декількох цілей, а для їх використання, в сутності повинна бути описана поведінка, з якою вона не пов’язана. Принцип полягає в розділенні масивних інтерфейсів на множину невеликих, але призначених для однієї задачі. Слідування принципу надає легкість в підтримці програмного забезпечення – зміни в коді інтерфейсу спричинять зміни тільки в тих сутностях, які безпосередньо використовують змінений функціонал.

Принцип інверсії залежностей найбільш всеохоплюючий з 5 SOLID-принципів. Він пропонує побудову архітектури системи на автономних, незалежних один від одного модулях, вся взаємодія між ними знаходиться на рівні абстракції. Такий підхід виявився неймовірно корисним для великих систем – зміна в одному модулі спричиняє модифікацію тільки в ньому самому та на рівні абстракції, безпосередньо не впливаючи на інші компоненти системи.

SOLID-принципи зараз вважаються класичними, а відповідність архітектури програмного забезпечення ним – кращою практикою. Вони допомагають уникнути багатьох проблем в процесі розробки будь-якої складності системи. Через свою простоту та універсальність вони підходящі для використання в рішеннях великої кількості задач, але існують виключення. Також часто системи слідують тільки декільком принципам, відмовляючись від інших, як надлишкових для своєї специфіки.

Архітектура розробленої системи відображення та аналізу даних відповідає принципу єдиної відповідальності – переважна більшість типів в ній мають лаконічну структуру та всі вони розроблені для одного відповідного

специфічного функціоналу.

Суть розробленої системи полягає в обробці даних технологічного процесу, тому перше питання, яке було вирішене в ході розробки архітектури даної системи – це визначення формату даних та їх джерела.

Існує декілька варіантів збору даних технологічного процесу на виробництві. Найпопулярнішими виступають два підходи, інші представляють собою їх комбінацію.

Перший підхід найчастіше використовується при модернізації технологічного процесу на виробництвах із застарілим обладнанням. Він полягає в переважно апартній складовій – збір даних відбувається шляхом підключення технологічного обладнання до окремих пристроїв. Наприклад, розповсюджена система «Диспетчер»[12] пропонує збір даних за допомогою різноманітних терміналів – від регістраторів цифрових та аналогових сигналів, до терміналів введення-виведення, які взаємодіють з оператором. Вони оснащені різними засобами передачі інформації. Найчастіше зустрічається використання стандартів Ethernet для передачі даних на сервер та інтерфейс RS-485 для взаємодії терміналів між собою. Термінали підключаються безпосередньо до обладнання технологічного процесу, що вирішує проблему відсутності необхідних інтерфейсів для збору та передачі інформації.

Інший підхід переважно використовується на виробництвах із більш сучасними пристроями. Останнім часом парадигма автоматизованого та автоматичного технологічного процесу все повніше впроваджується на виробництвах, оскільки її переваги незаперечні – підвищення якості, продуктивності, безпечності технологічного процесу, тощо. Ще однією її вагомою перевагою виступає оснащення переважної більшості сучасних пристроїв інтерфейсами, які підтримують популярні протоколи передачі даних. В контексті задачі, що розглядається, це сильно спрощує процес збору даних – залишається тільки сконфігурувати локальну мережу для підключення приладів до серверу.

Наведені вище два найпопулярніших підходи принципово відрізнаються

один від одного, проте вони також мають спільні аспекти. Основний з них – в результаті вся необхідна інформація технологічного процесу зберігається на сервері. Часто під терміном «сервер» мають на увазі деяку масивну робочу станцію з високою обчислюваною потужністю та великим об'ємом внутрішньої пам'яті. Але в подібних системах, особливо, якщо вони впроваджуються на невеликих підприємствах, серверною стороною може виступати і звичайний комп'ютер(за відповідності ним мінімальним технічним вимогам), а замість бази даних інформація може зберігатись в звичайній таблиці створеній в програмі Microsoft Exel.

На сьогоднішній день, технологічний процес виробництва шоколаду не має єдиного стандартизвованого підходу до збору та збереження інформації, для кожного конкретного виробництва методи індивідуальні та спецефічні.

Проте, оскільки дані технологічного процесу виготовлення темного шоколаду – це переважно інформація з дачиків та відповідей автоматизованих установок, їх необроблене представлення слабко адаптоване для інтерпретації людиною. Тому, ефективно та вигідно їх збирати та зберігати в форматі CSV. Як було описано в розділі 1, такий формат є універсальним та один із найзручніших для передачі ним інформації для обробки в до ігшої системи.

В результаті аналізу наведених вище особливостей збору та зберігання даних технологічного процесу виготовленя шоколаду було визначено, що система набуде найбільших властивостей універсальності при обробці вхідних даних, які вона отримує в форматі CSV, так як в великій частині випадків вона саме в такому форматі зберігається на сервері, а в іншій – інформація, що зберігається в таблицях може бути легко конвертована в даний формат та навпаки(кожен рядок файлу представляє собою рядок таблиці, дані комірок розділені комою).

Також окремим модулем системи виступає модуль підключення до сховища. В ньому визначені формати запиту до бази даних, налаштування та підтримка зв'язку з базою даних функціонал посилання запитів та прийом відповідей на них. В рамках даного проекту конкретна реалізація цього модулю

опускається, оскільки його реалізація сильно залежить від підходу до збереження даних, обраного на конкретному виробництві, де впроваджується дана система, а в рамках обраного підходу - вона досить стандартна та може бути реалізована безпосередньо на виробництві.

3.1 Опис структури та поведінки системи

Структурна та функціональні схеми даної системи наведені в ілюстративному матеріалі проекту. Нижче наведений алгоритм роботи системи.

Оскільки результат роботи системи набуває найбільшої актуальності при відображені даних в режимі близькому до реального часу, її алгоритм включає періодичний запит даних, що робить процес її роботи ітеративним. Один повний цикл роботи системи описується наступним чином.

3.1.1 Вхідні дані

Як було визначено, формат вхідних даних – CSV. В класичному представлені дані розділені комою. Початком кожної ітерації слугує отримання двох наборів вхідних даних. Перший набір представляє дані технологічного процесу – наприклад, маса какао-крупки після обробки, об'єм масла какао, який потравляє в суміш, температура на валах млинів, тощо. Другий набір даних являє собою набір налаштувань, з використанням яких будуть оцінюватись дані технологічного процесу. Інформація про налаштування необхідна системі тільки разово, на початку роботи.

З точки зору кінцевого користувача, більш зручним було б рішення задання налаштувань безпосередньо в коді самої системи або за допомогою користувацього інтерфейсу. Але проти цього виступає два фактори.

По-перше, такий підхід із зовнішнім вказанням налаштувань робить систему більш гнучкою до специфіки конкретного технологічного процесу виготовлення шоколаду. Він надає можливість більш точно оброблювати дані

відповідно до обраного рецепту з індивідуально встановленими пропорціями та налаштуваннями параметрів технологічного процесу.

По-друге, після визначення рецепту приготування шоколаду, на виробництві він дуже рідко змінюється, тому нема потреби в постійному швидкому доступі до налаштувань обробки інформації, це робить користувацький інтерфейс більш лаконічним та зрозумілим.

3.1.2 Валідація вхідних даних

Первинна обробка вхідних даних виконується в окремому сервісному модулі системи. Він відповідає за парсинг вхідних даних, перевірку їх присутності та адекватності. При перевірці на присутність даних, в разі пропусків, вводяться додаткові перевірки, які допомагають локалізувати несправність. Наприклад, якщо відсутнє тільки одне значення з ряду – ймовірно, що саме з приладом/терміналом/датчиком сталась несправність, а в разі, якщо в «комірці», де має знаходитись число, знаходиться літерал – більш ймовірно, що несправність виникла при формуванні файлу на сервері.

Такі базові перевірки необхідні, оскільки вони виступають в ролі так званого «health-check» для системи та в разі відмови будь-якого приладу, за яким ведеться спостереження, про це відразу буде повідомлено. Також вони необхідні для роботи системи. Оскільки немає сенсу оброблювати некоректні дані, задача системи буде зводитись до повідомлення про несправність та очікування нових даних.

3.1.3 Формування сутностей даних та налаштувань

Після успішної валідації вхідних даних, їх необхідно оформити у відповідні структури для подальшого використання в алгоритмі. Вхідні дані формуються в об’єкти двох сутностей – це сутність даних технологічного процесу Parameter та сутність налаштування Setting.

Вони лаконічні та представлють собою тільки структуру, в яких збергаються дані, ніяким функціоналом вони не наділені та виступають тільки в ролі «постачальників» даних.

3.1.4 Аналіз параметрів технологічного процесу

Етап процесу аналізу даних полягає у визначенні границь, в межах яких знаходиться кожний оброблюємий параметр, для нього програмно визначений окремий модуль. Границі можуть бути встановлені за замовчуванням, або взяті з налаштувань, якщо вони там були вказані. В результаті, для кожного параметру встановлюється окреме поле перелічувального типу в одне з наступних значень:

- параметр знаходиться в границях норми;
- параметр перетнув границю попередження;
- параметр перетнув аварійну границю;
- параметр перетнув аномальну границю.

Також, кожний параметр інкапсулює окремий об’єкт типу Calculation, який несе в собі всі отримані попередні значення параметру, та використовується для підрахунків середнього арифметичного значення параметру. Ці дані необхідні для подальшого аналізу.

3.1.5 Відображення результатів аналізу

На цьому етапі відбувається відображення даниих на користувацькому інтерфейсі у вигляді графіків двох типів – гістограми та лінійного графіку. Приклад сформованого зображення на моніторі користувача зображений на рисунку 4.

Вікно програми містить набір панелей, на яких зображені індивідуальні гістограми для кожного параметру, та лінійні графіки, які відображають попередні значення параметру, значення якого вийшло з норми. В данному прикладі – значення маси цекру потрапило в область попередження, а значення об’єму олії какао – в область небезпеки, тому справа від гістограм даних

параметрів розміщені додаткові лінійні графіки.

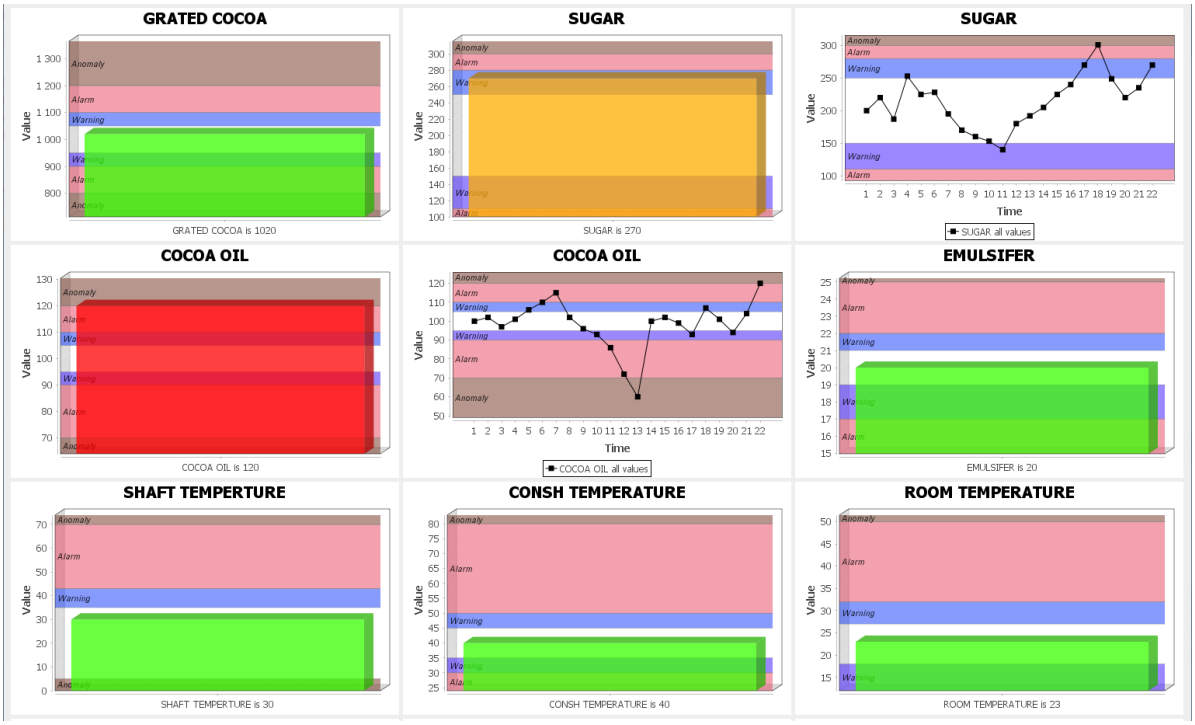

Рисунок 4 – Приклад вигляду користувацького інтерфейсу

Для відображення результатів аналізу програмно також було виділено окремий модуль, який утримує в собі наступні компоненти:

- компонент, який конвертує дані для необхідного вигляду, при побудові графіків на їх основі;
- компонент, який відповідає за дизайн графіків;
- компонент, який відповідає за створення та налаштування графіків;
- компонент, який формує всі необхідні графіки на одній панелі.

Для підвищення зручності обробки людиною зображених графіків, кожен з параметрів зображений на окремій гістограмі, яка окрім незначних налаштувань (таких як колір заднього фону, назва графіку, тощо) має додаткові, більш специфічні для кожного параметру.[14]

Перш за все, це виділення кольором на кожній гістограмі заданих або стандартних границь. Для кожного параметру вони індивідуальні. Виділяються границі шляхом зафарбовування відповідних зон на вертикальній осі, вздовж якої зображений графік. Можливе виокремлення трьох кольорів :

– виокремлення зони попередження (Warning) синім кольором;
– виділення зони небезпеки (Alarm) прозоро-рожевим кольором;
– виділення зони аномалії (Anomaly) прозоро-коричневим;

Рамки визначеної області нормальних значен додаткови кольором не виділяються – вони залишаються білими, так само, як і задній фон самого графіку. Враховуючи, три кольори, якими виділяються області, та змінний колір стовпчика значення додаткове виділення нормальної області створювало б занадто контрасне зображення, на якому було б важко сфокусувати увагу. На Рисунку 5 зображено приклад відображення параметру – маса какао-крупки. В даному прикладі, дані, що були передані знаходяться в границях норми, це видно з графіку, оскільки стовпчик значення зеленого кольору, а показник параметру залишається на білому фоні.

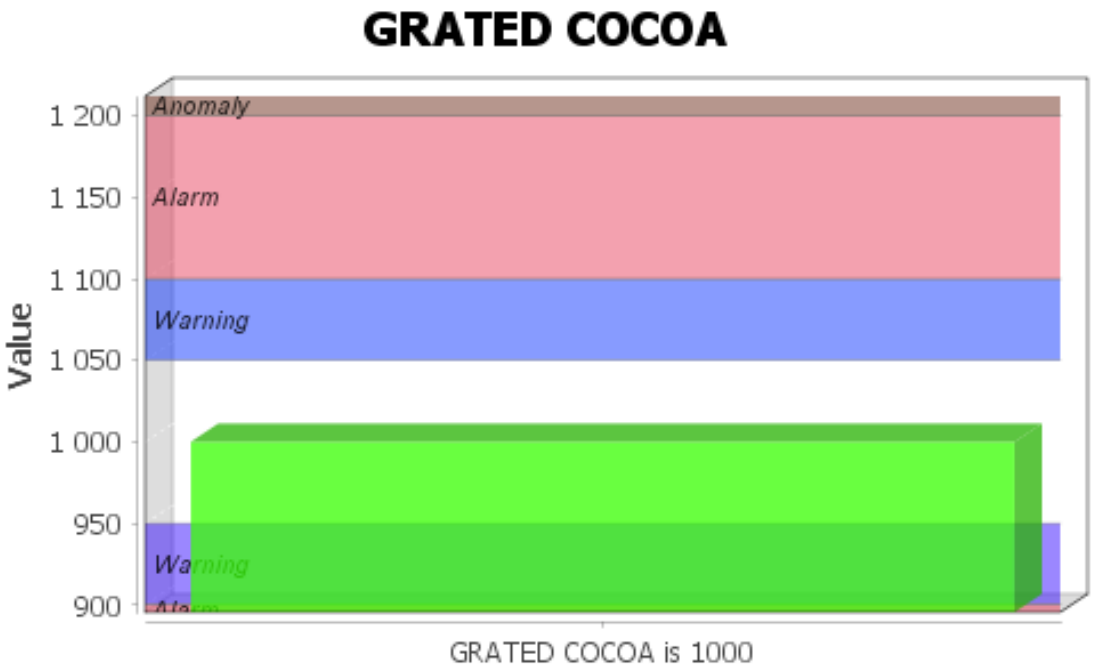

Рисунок 5 – Приклад відображення показнику, який знаходиться в рамках норми

Для кожного графіку також додатково налаштований надпис, який засвідчує значення параметру. Кольори, що відмічають зони, які були визначені для кожного параметру мають налаштовану прозорість. Завдяки цьому, вони не

перекривають значення параметру. Це можна помітити на Рисунку 6, де відображено значення який знаходиться в зоні попередження. Оскільки значення перевищило за границі норми, колір стовпчику також змінився на помаранчевий.

Як зображено на, Рисунку 6, позначення границь все ще зображені та не перекриваються стопчиком, що відображає значення параметру. Це надає свої переваги – наприклад, допомагає швидко візуально оцінити, насільки сильно отримане значення відхиляється від норми, або тої чи іншої границі.

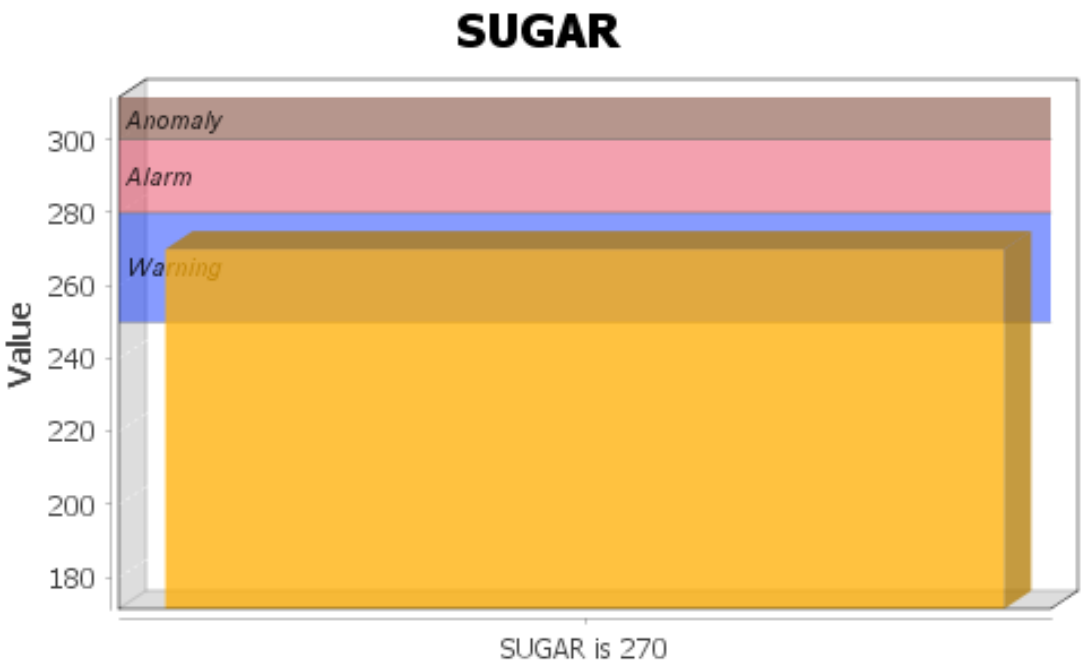

Рисунок 6 – Відображення параметру, який знаходиться в зоні попередження

При відображенні параметрів, з метою досягти лаконічне та зрозуміле на інтуїтивному рівні зображення значень параметрів, всі графіки, розміщені на панелі мають один й той самий розмір, допомагає не акцентувати увагу оператора на одному параметрі та точніше слідкувати за всіма важливими характеристиками.

Недоліком такого підходу виступає той факт, що системою оброблюються параметри різного характеру, тому нормальні значення параметрів та їх границі можуть суттєво відрізнятись між собою. При великих значеннях границь, або

якщо вони близько за значеннями розташовані одна до оної, візуально стає складно оцінити значення параметру відносно його норми та інших границь. Приклад відображення такого параметру зображений на Рисунку 7. На графіку зображено отримане значення маси како крупки та встановлені границі. В результаті аналізу, було визначено, що значення параметру перевищило границю попередження та небезпеки, але не перевищило границю аномалії.

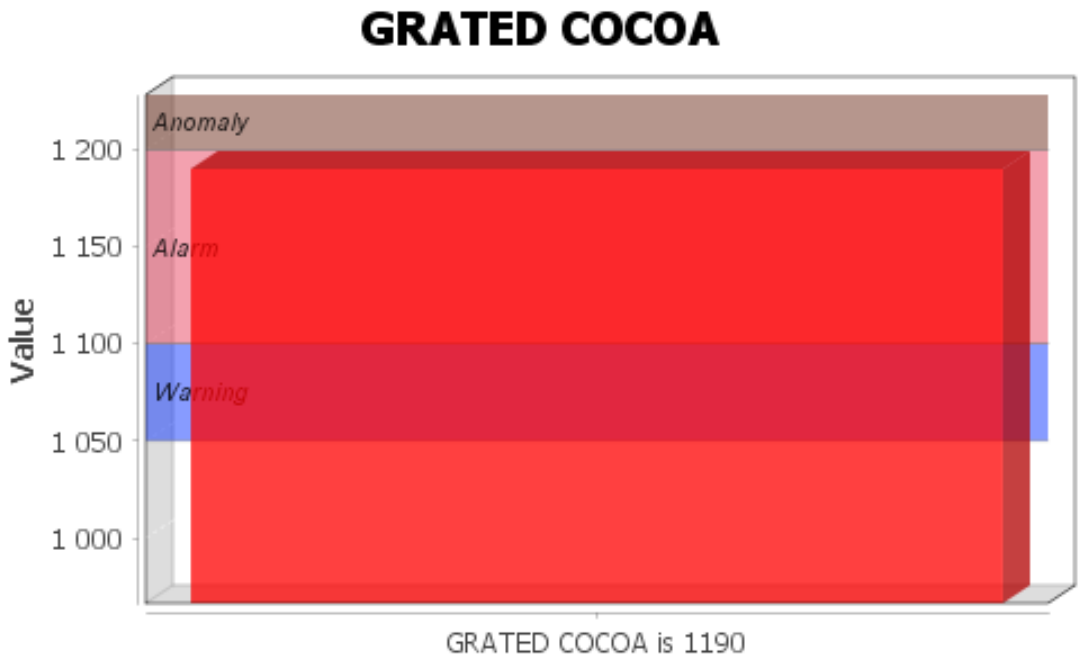

Рисунок 7 – Відображення параметру з границями, щільно розташованими між собою за значеннями

Оцінити таку гістограму в первинному вигляді допомагає тільки колір стовпчику, що відображає значення параметру. Для значення в зоні аномалії, стовпчик значення набуває фіолетового кольору, а оскільки на отриманому графіку він червоний, стає зрозуміло, що значення знаходиться в області небезпеки.

Хоча кольорове забарвлення саме по собі надає можливість однозначно оцінити значення параметру, а вказане під графіком значення допомагає в цьому, таке відображення все ще не є зручним для людського ока.

Для уникнення хибного сприйняття графіків, та з метою зробити

відображення отриманих параметрів більш прозорим для оцінки оператором, виведені на екран гістограми підтримують інтерактивність.

Таким чином, отриманий графік, зображений на Рисунку 7 можна «розтягнути», тобо збільшити для обраних начень числової осі і він набуде вигляду, зображеного на Рисунку 8.

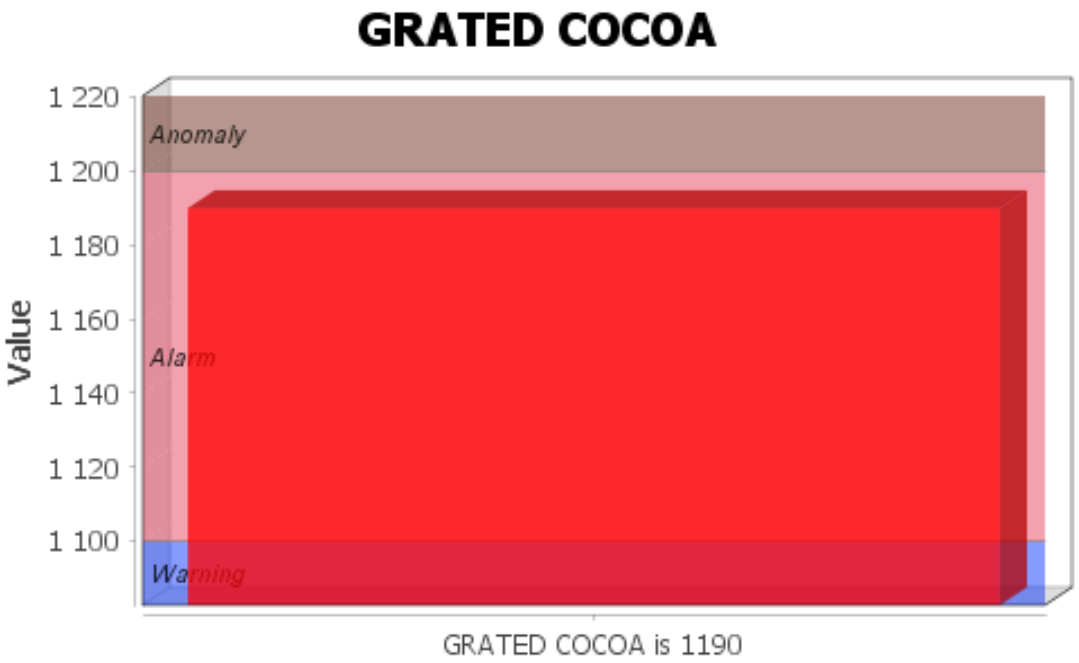

Рисунок 8 – Графік зі збільшеною числовою віссю

При такому вигляді гістограми, відображення значення параметру та його відношення до границь областей попередження та аномалії стає наглядним та більше не викликає сумнівів.

Також, для тих параметрів, значення яких вийшло за границю попередження формуються лінійні графіки, які відображають тенденцію попередніх значень, які були отримані. Приклад такого графіку зображений на Рисунку 9.

На ньому можна побачити, що останнє значення (крайнє зліва) отриманого показника (в даному випадку – об’єм масла какао) знаходиться на межі між небезпекою та аномальною поведінкою. Хоча, значення та результат його обробки в продальшому підлягає додатковому аналізу та збереженню, візуальне

відображення статистики його зміни може бути також корисним.

При цьому відображати останні попередні значення кожного параметру, незалежно від результатів його обробки, суттєво захаращувало б користувацький інтерфейс, та не несло б практичної користі.

Розраховане середнє арифметичне значення показника на графічному інтерфейсі не відображається, так як лінійний графік дає можливість візуально оцінити його приблизне значення, а додаткова лінія тільки відволікатиме увагу оператора.

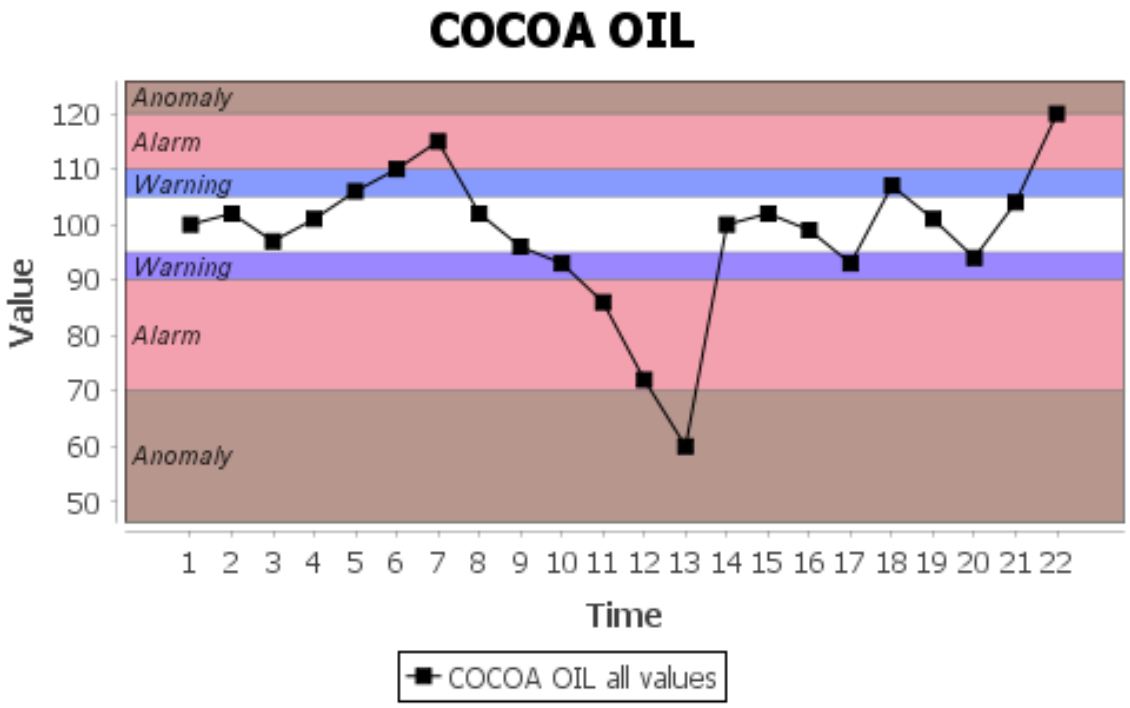

Рисунок 9 – Приклад відображення лінійного графіку

Графічний інтерфейс користувача також надає можливість взаємодії із зображеним графіком. Більшість налаштувань пов’язана з зовнішнім виглядом графіку. До них відносяться:

– налаштування назви графіку (редагування назви, обраного шрифту та кольору);
– налаштування вісей графіку (редагування мінімального та максимального значень числової вісі, шрифту та кольору відміток для обох осей, редагування фону вісей, тощо);

– загальні налаштування (наприклад, встановлення зображення на задній фон графіку, тощо).

Також кожний графік має опцію автоматичного масштабування, проте її не рекомендується використовувати. В процесі моделювання кожного графіку, з урахуванням поточного значення параметру, та значеннями границь відповідних областей, встановлених для нього, булі встановлені індивідуальні мінімальне та максимальне значення на числовій осі. Це може привести до нехтування деяким простором вісі, але при цьому допомагає більш вдало відобразити значення параметру.

Результат роботи цього механізму можна помітити на Рисунку даного пункту – числова вісь починається не з нуля а з різних значень, в залежності від параметру, що відображається.

При використанні опції автоматичного масштабування графік перебудується, числова вісь буде починатись з нуля та закінчуватись декількома значеннями після значення параметру. Такий підхід часто не є зручним для відображення, оскільки, більшу частину гістограми буде займати стовпець значення монотонного кольору, та в частині випадків будуть зображені «обрізані» зони визначених для нього границь. Таке відображення буде надавати неповну інформацію та бути менш зруним для візуальної оцінки параметру.

3.1.6 Обробка параметрів, значення яких поза області норми

Після первинного аналізу та відображення отриманих та програмно згенерованих параметрів, формується окремий набір параметрів та результатів їх обробки із тих параметрів, значення яких вийшли з області нормальних значень – тобто, перетнули границю попередження, небезпеки або аномалії.

Отриманий набір даних додатково аналізується на предмет виявлення взаємозв’язку між параметрами та встановлені сили такого зв’язку. Очевидно, що не кожний параметр буде пов’язаний з кожним – наприклад, від температури, яка підтримується всередині кошн-машини не залежить маса како-крупки, яка

подається на обробку. Проте існують пов’язані параметри – наприклад, від тиску, що створюється між валами млина залежить температура, яка встановлюється на зовнішній площі валів, в цього напряму залежить температура шоколадної маси, яка між ними проходить, і так далі.

Визначення такого взаємозв’язку між параметрами зробить технологічний процес прозорішим та прогнозованішим. Для цього в системі використовується коефіцієнт кореляції Пірсона. Результат його розрахунку визначає, наскільки зміна значення одного параметру пропорційно вплине на інший параметр, надаючи відповідь у вигляді коефіцієнту, значення якого знаходяться в інтервалі від 0 до 1.

Використання коефіцієнту кореляції Пірсона потребує виконання трьох наступних вимог:

- значення параметрів, які оцінюються тають бути нормально розподіленими;
- значення параметрів мають бути виміряні метричною шкалою, або шкалою відношень;
- параметри, що оброблюються мають мати однакову кількість значень.

Можна сказати, що для оброблюємих параметрів всі вимоги виконуються, оскільки їх значення вимірюються метричною шкалою, при первинному аналізі поточне значення для кожного з параметрів зберігається, що формує набір даних однаковий за розміром для кожного параметру та з часом розподілення їх значень стає все більш близьким до нормального.

Формула, що була використана для розрахунку коефіцієнту кореляції Пірсона має наступний вигляд:

$$r_{xy} = \frac{\sum(x_i - \bar{x}) * (y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 * \sum(y_i - \bar{y})^2}}, \qquad (3.1)$$

де x, y – значення параметрів, які оброблюються;

$\bar{x}$ – середнє арифметичне значення першого параметру;

$\bar{y}$ – середнє арифметичне значення другого параметру.

Дані, які для цього використовувались зберігаються в самій сутності Parameter , а точніше – в інкапсульованій ній сутності Calculation. Функціонал обчислення коефіцієнту кореляції був винесений до окремої сутності з метою збереження принципу єдиної відповідальності, який згадувався вище.

Результати розрахунків для кожної пари параметрів представляють значення коефіцієнту в межах від 0 до 1. Нуль означає, що параметри, на даній вибірці не пов’язані між собою, а зміна одного зних не буде корелювати за зміною іншого. Одиниця трактується як відношення між параметрами за лінійним законом.

Також, результуюче значення коефіцієнту може бути від’ємним. Загальне трактування значення коефіцієнту визначає:

- при значенні r_{xy} менше 0,3 сила кореляції вважається слабкою, та нею нехтують при подальших розрахунках;
- при значенні r_{xy} в інтервалі від 0,3 до 0,7 сила кореляції вважається помірною, або помітною;
- при значенні r_{xy} вище 0,7, сила кореляції вважається високою.

Отримані результати не зображуються на графічному інтерфейсі, оскільки нема необхідності їх відслідковувати безпосередньо в процесі роботи технологічного процесу, а несуть більш статистичний характер.

3.1.7 Збереження даних параметрів та результатів їх обробки

Оскільки вхідні дані, які були збережені в об’єктах сутності Parameter зберігаються віддалено на сервері, необхідність всі з них зберігати повторно відсутня. Але набуває сенсу збереження даних, які потрапили в область попередження, небезпеки або аномалії для ведення статистики та подальшого їх аналізу.

Тому, набір даних, який був сформований на попередньому кроці форматується в зручний для людини вигляд та записується в окремий файл з

роширенням txt. Кожен запис вихідного файлу несе собою вичерпну інформацію про стан параметру, який було визначено протягом даної ітерації, наприклад, час запису, значення параметру, значення границі, яку він перетнув(якщо така присутня), його середнє арифметичне значення, та інше.

3.2 Основні модулі та класи

В процесі розробки даної системи, її архітектура, яка була визначенна перед початком написання коду, зазнавала змін, які в результаті зробили її оптимальною та збалансованою основою системи.

Загальна структура класів та взаємозв’язків між ними в системі може бути умовно розбита на окремі сервіси в залежності від функціонального призначення, яке несе кожен окремий клас. До такого результату привело слідування принципу єдиної відповідальності, який було описано вище та ідеї розбиття системи на множину мікросервісів.

В результаті наслідування першого принципу, сутності, якими оперує система, були детально оцінені з точки зору функціональних можливостей, які вони повинні надавати та були максимально розбиті на різні лаконічні та прості типи.

Не дивлячись на те, що такий підхід суттєво збільшив кількість сутностей, якими оперує система, та зробив зв’язки між ними більш заплутаними, він наділяє систему рядом переваг, таких як:

- кожна сутність несе собою тільки невелику частину функціоналу, що спрощує її використання в коді, оскільки робить її простою та однозначною;
- механізм відлагодження та модифікації існуючого функціоналу також стає простим (наприклад, при виникненні помилки, процес відслідковування її шляху крізь зв’язки між сутностями завдяки строгому визначенню границь сутностей скорочується, процес локалізації несправності або логічної помилки стає прозорим);

– механізм впровадження нового функціоналу потребує мінімальну кількість змін у вже написаному коді.

Впроваджена ідея розбиття цільної системи на множину сервісів також зробила її більш гнучкою та змін та зручною в процесі розробки. Були виділені окремі блоки сутностей та функціональних можливостей, об’єднані спільною ціллю. В даній системі сформувались наступні модулі:

– модуль підключення та роботи з базою даних;
– модуль аналізу даних;
– модуль відображення даних на користувацькому інтерфейсі;
– модуль збереження даних.

Кожний модуль має свою внутрішню структуру, а функції, визначені для кожного з них не потребують функціонального втручання інших модулів. Це обежує взаємодію між окремими модулями , спрощує процес підтримки та розширення системи.

Такий підхід дозволяє вдало використовувати один з принципів розробки, який надає Java як об’єктно-орієнтована мова програмування – інкапсуляцію.

Під терміном «інкапсуляція» мається на увазі принцип, згідно з яким, всі дані, сутності, типи, тощо мають бути доступні тільки тій області програми, яка безпосередньо їх потребує, а від іншої частини вони мають бути приховані.

Такий принцип реалізується за допомогою багатьох підходів проектування програмного забезпечення. Основні з них, які були використанні в процесі розробки даної системи – це обмеження області видимості об’єктів за допомогою маніпуляції модифікаторами доступу та формування даних та функції в окремі сутності та сервісні класи (тобто, класи, які спроектовані тільки для надання функціоналу статичними методами та не потреують створення об’єкту даного типу).

Також, в процесі проектування системи були використані декілька шаблонів проектування програмного забезпечення.[13]

При проектуванні підключення до бази даних був використаний шаблон Однак(в перекладі - Singleton). Цей шаблон часто використовується для

сутностей, які встановлюють зв’язок із розподіленим ресурсом між клієнтами. Використання даного шаблону відносно деякого класу гарантує, що в програмі буде створений один і тільки один об’єкт даного класу із глобальним доступом до нього. Механізм, яким це забезпечується полягає в приховуванні конструктору об’єкту для доступу ззовні та створенні публічного методу, який перевіряє, чи був створений об’єкт класу раніше і, якщо так – повертає його, якщо ні – викликає конструктор, та повертає створений об’єкт.

Також, деякі сутності даної системи, хоча й несуть собою специфічні набори функцій, які можуть бути визначеними як «єдина відповідальність», оперують великою кількістю параметрів. Через це постає питання вдалого їх отримання та проектування цього механізму всередині сутностей.

Існує багато рішень цього питання. Найпростіше полягає у створенні нового конструктору з усіма необхідними параметрами. Але такий підхід вважається поганою практикою через те, що при створенні об’єкту, вказувати велику кількість параметрів дуже незручно та в процесі розробки стає легко зробити логічну помилку. Також, в разі розширення сутності, можуть з’явитись нові необхідні параметри, частина старих може стати опціональною, тощо, а такий строгий контракт створення об’єкту приведе до значних змін у вже написаному коді та неоднозначності використання сутності.

Іншим простим рішенням може бути створення декількох конструкторів в залежності від параметрів, які необхідні кожному конкретному об’єкту. Такий підхід приводить до суттєвого збільшення класу, велика частина коду якого практично дубльована.

Тому в процесі проектування таких сутностей був використаний шаблон Будівник (в перекладі - Builder). Ідея шаблону полягає в поетапному формуванні об’єкту за допомогою проміжкових методів та завершення формування деяким термінальним методом (часто має назву build та не має параметрів). Такий підхід спрощує формування складних об’єктів, робить вигляд коду більш очевидним та робить тип гнучким для модифікації.

3.2.1 Основна сутність системи – Parameter

В архітектурі даної системи об’єкти типу Parameter виступають в ролі однієї з основних інформаційних одиниць. Кожен об’єкт створється після валідації даних та перевірки значень на адекватність, значення що в ньому зберігаються придатні для обробки.

Кожен об’єкт зберігає чотири значення. Перше поле – індивідуальний для кожного об’єкту числовий ідентифікатор – id. Поле доступне тільки для зчитування, задається воно при створенні об’єкту типу Parameter. Друге поле – value, значення параметру, отримане із вхідних даних.

Третє поле представляє об'єкт окремої сутності CheckedParameter, який формується після первинного аналізу. Четверте поле – об'єкт іншої складної сутності Checker.

Формування об'єкту типу Parameter - досить складний процес. При створенні об’єкту, спершу ініціюється об’єкт типу Checker. Цей об’єкт виступає в ролі зв’язку між об’єктом типу Parameter та множиною сутностей та класів, які використовуються для аналізу даних.

Після цього викликається метод setValue, який присвоює нове значення поля, яке зберігає значення кожного параметру, та створює об’єкт сутності CheckedParameter для нового значення атрибуту(в процесі створення, параметр підлягає перевіркам, відповідно до заданих налаштувань).

Ззовні для зміни доступне тільки значення value, зміна якого знову зініціює аналіз параметру та об’єкт, що зберігає результат обробки буде заміщений новим. Поля сутностей типів CheckedParameter та Checker не мають методів для зовнішнього встановлення або зчитування, їх значення повністю інкапсульовані в об’єкті типу Parameter.

В результаті, отримана сутність є дуже лаконічною – для програми вона представляє собою тільки значення та його ідентифікатор, а всі інші методи, які з нею пов’язані, але виходять за її відповідальність винесені в інші допоміжні

сутності та класи.

3.2.2 Клас Checker та сутності, які з ним пов’язані

Для кожного параметру, при ініціалізації створюється новий об’єкт типу Checker, який існує протягом всього часу існування об’єкту параметра. Даний об’єкт можна вважати сервісним, оскільки він не несе в собі бізнес-логіки, а тільки виступає в ролі зв’язу між даними та частиною функціоналу системи.

Кожен об’єкт має три приватні поля :

– об’єкт типу Setting, який несе собою налаштування границь параметру;
– об’єкт типу Calculation. В ньому для заданого параметру зберігаються його попередні значення та функціонал обрахунку середнього арифметичного значення;
– об’єкт типу BorderCheck. В ньому виконується обробка даного параметру відповідно до заданих налаштувань.

Сам об’єкт типу Checker має тільки один публічний метод , який викликає відповідні методи об’єктів calculation та borderCheck (в яких протікає процес первинного аналізу кожного значення), формує об’єкт типу CheckedParameter та повертає його.

Розглянемо детальніше об’єкти calculation та borderCheck. Вони вже безпосередньо представляють собою частину бізнес-логіки. Кожен з кроків аналізу в архітектурі даної системи розбитий на два типи сутностей. До першого формату відносяться сутності типів Calculation та BorderCheck – вони надають тільки функціонал, за домогою якого виконується аналіз, та зберігають в собі тільки дані, необхідні для нього. В результаті аналізу створюється відповідна сутність іншого формату. Для даних типів – це сутності типів CalculationResult та BorderCheckResult. Вони несуть собою тільки результати проведеного аналізу та методи, які надають до них доступ ззовні класу.

Впроваджений підхід має багато переваг. Основними виступають наступні:

– сутність результатів обробки досить слабко пов’язана із сутністю

безпосередньо обробки. Таке архітектурне рішення дозволяє легко впроваджувати нові методи обробки та змінювати існуючі – змінам буде підлягати тільки одна сутність, а задача зведеться до приведення відповіді методів до єдиного формату;

– так само слабко пов’язана сутність типу Checker з сутностями, які виконують аналіз параметру. Переваги мають більший масштаб, хоча того ж характеру – висока гнучкість модулю;
– подальша обробка значеня параметру потребує тільки результатів їх аналізу, всьому іншому коду не потрібен доступ до функціоналу їх обрахунку, його приховання також допомагає уникнути логічних помилок в процесі розробки.

В результаті єдина функція сутності типу Checker - метод check, який ініціює процеси аналізу та формування результатів, створює та повертає комплексний об’єкт типу Checked Parameter до параметру, в якому він був викликаний. Функціонал та мета кожної сутності строго визначені та розділені, а від зовнішнього алгоритму роботи системи вони приховані.

3.2.3 Сутність CheckedParameter

Об’єкт даної сутності включає в себе всі результати первинного аналізу значення параметру, сам об’єкт параметру та його налаштування. Всі поля структури приватні та мають публічні методи для читання. Задаються вони один раз – при створенні об’єкту, в конструкторі.

Сутність спроектована з використанням шаблону проектування «Будівник». Це виражено в тому, що всередині неї описан ще один статичний клас CheckedParameterBuilder. Його тіло зображене на Рисунку 10.

Мета класу – задати послідовне, ітеративне створення об’єкту типу CheckedParameter через використання методів, які встановлюють обрані параметри, та закінчуючи термінальним методом build, який повертає створений об’єкт.

```
public static class CheckedParameterBuilder {

    private CheckedParameter checkedParameter;

    public CheckedParameterBuilder() {
        checkedParameter = new CheckedParameter();
    }

    public CheckedParameterBuilder withParameter(Parameter param) {
        checkedParameter.param = param;
        return this;
    }

    public CheckedParameterBuilder withSetting(Setting setting) {
        checkedParameter.setting = setting;
        return this;
    }

    public CheckedParameterBuilder withBorderCheckResult(BorderCheckResult borderCheck) {
        checkedParameter.borderCheck = borderCheck;
        return this;
    }

    public CheckedParameterBuilder withCalculation(CalculationResult calculation) {
        checkedParameter.calculation = calculation;
        return this;
    }

    public CheckedParameter build() {
        return checkedParameter;
    }
}
```

Рисунок 10 – Побудова об’єкту CheckedParameter за допомогою шаблона «Будівник»

Оскільки кожний з методів, окрім термінального повертає сам об’єкт CheckedParameterBuilder, це дає можливість використовувати іншу впроваджену практику програмування – ланцюговий виклик методів.

```
public CheckedParameter check(Parameter param) {
    return new CheckedParameter.CheckedParameterBuilder()
            .withParameter(param)
            .withSetting(setting)
            .withCalculation(calc.calculate(param.getValue()))
            .withBorderCheckResult(borderCheck.checkBorders(param.getValue()))
            .build();
}
```

Рисунок 11 – Створення об’єкту типу CheckedParameter

На Рисунку 11 зображено тіло основаного методу сутності – об’єднання звернень до необхідних сутностей, ствоння об’єкту та ланцюговий виклик

функцій зробили, при великій кількості виконуваних дій, код лаконічним, гнучким та легко читаємим.

Сутність представляє собою весь необхідний набір даних для відображення на користувацькому інтерфейсі та подальшого аналізу параметру, в разі його невідповідності нормам.

Також, з метою полегшення розробки та підвищення читаємості коду був створений додатковій перелічувальниій тип BorderCheckStatus, який включає в себе наступні поля:

- Normal;
- Warnning;
- Error;
- Anomaly.

Значення цього типу характеризує, в якій області знаходиться значення оброблюємого параметру. Воно визначається на етапі аналізу параметру відносно заданих границь, та використовується переважно в модулі відображення результатів обробки та в процесі їх збереження.

3.2.4 Модуль відображення даних

Модуль відображення даних даної системи включає в себе чотири основні сутності та їх взаємодію між собою. Кожна з цих сутностей також була спроектована відповідно до принципу єдиної відповідальності, та вони представляють собою переважно методи формування та відображення графіків. Цими сутностями виступають:

- DataSetManager;
- ChartDesign;
- FormChart;
- DisplayResults.

Вхідними даними цього модулю виступає набір об’єктів типу CheckedParameter , а на виході відображаються сформовані графіки. Розглянемо

детальніше кожен клас.

Тип DataSetManager виповідає за приведення значень параметру до формату, який буде використовуватись при побудові графіку. В даній системі значення параметрів відображається за допомогою гістограм та лінійних графіків, вони є категоріальними графіками, тому дані кожного параметру переформовуються в структуру типу CategoryDataset за допомогою відповідних методів – createDataset та createDatasetForLineChart .

Клас ChartDesign представляє собою великий набір функцій, які задають всі необхідні налаштування відображення графіку. Серед них присутні:

- метод, який задає задній фон панелі графіку;
- метод, який задає задній фон графіку;
- метод, який обраховує область видимості для кожного графіку. В залежності від значення параметру, який відображається та визначених для нього границь, для кожного графіку встановлюються мінімальне та максимальне значеня осі ординат;
- група методів, яка для кожного графіку визначає, за якими значеннями (границями) розміщена кожна із вказаних областей, встановлює налаштування забарвлення їх відповідними колькором та додатковим надписом;
- метод, який надає стовпчику гістограми, який відображає значення параметру відповідного кольору, в залежності від області, в якій розміщується значення;
- метод, який встановлює значення легенди;
- метод , який налаштовує товщину лінії, формує позначки на графіку, колір лінії для лінійного графіку;
- термінальні методи, який поступово викликають всі перелічені вище функції та повертають об’єкти графіків.

Клас спроектований таким чином, що кожен метод виконує тільки одну функцію, при цьому максимально уникаючи дублювання всередині себе да з іншими методами.

Сутність FormChart є, в деякому сенсі, термінальною в процесі створення графіку. При створенні об’єкту цього типу, він потребує об’єкти CheckedParameter та DataSetManager, тобто результати обробки параметру та його значення в форматі, придатному для побудови графіку.

Мета цієї сутності полягає у єдиному публічному методі, який в ній описаний – createBarChart. В даному методі відбувається створення об’єкту графіку (типу JFreeChart) зі стандартними налаштуваннями (назва графіку, підписи осей координат та категорій, задання типу графіку, та інше). Після цього, створюється об’єкт сутності ChartDesign, де конфігуруються спецефічні налаштування для кожного графіку.

В результаті, функція повертає вже сконфігурований об’єкт графіку, який готовий для відображення.

Тип DisplayResults оперує списком об’єктів графіків, які були створені та сконфігуровані в сутності FormChart. Його основний функціонал, який описаний в методі formChartsOnFrame, полягає в розміщенні в рамках одного вікна графічного інтерфейсу всіх необхідних графіків.

Для цього, в рамках контексту бібліотеки JFreeChart, ітеративно для кожного об’єкту графіку створюється панель, на якій він розміщений (об’єкт типу ChartPanel), задаються її фіксовані розміри та вони розміщуються на площі одного вікна.

3.2.5 Сутність LinkedParameters

Після аналізу вхідних даних відповідно до заданих границь формується ще один набір об’єктів типу CheckedParameter, який складається з тих, які перетнули, як мінімум, границю попередження. Вони підлягають додатковій обробці на предмет виявлення закономірності між несправностями, які були зафіксовані.

Як зазначалось вище, обробка відбувається за допомого розрахунку коефіцієнта кореляції. Дані для цього зберігаються в об’єктах типу

CalculationResult, які інкипсульовані в об’єктах типу CheckedParameter.

Обрахунок коефіцієнту відбувається в окремій сутності типу LinkedParameterCalculation. Вона містить набір статичний методів та список об’єктів LinkedParameters. Основний метод в ній – calculateLinkStrength. Його реалізація зображена на Рисунку 12.

```
private static Float calculateLinkStrength(Float average1, Float average2,
        List<Integer> val1, List<Integer> val2) {
    Float numerator = 0F;
    for(int i =0; i< val1.size();i++) {
        numerator +=(val1.get(i)-average1)*(val2.get(i)-average2);
    }
    double denumerator = 0F;
    Float sumVal1 = 0F;
    Float sumVal2 = 0F;
    for(int i =0; i< val1.size();i++) {
        sumVal1 += (val1.get(i)-average1)*(val1.get(i)-average1);
        sumVal2 += (val2.get(i)-average2)*(val2.get(i)-average2);
    }
    denumerator = Math.sqrt(sumVal1*sumVal2);
    return (float) (numerator/denumerator);
}
```

Рисунок 12 – Тіло функції обрахунку коефіцієнту кореляції

Дані для методу видобуваються з об’єктів типу CalculationResult в публічному методі класу LinkedParameterCalculation – checkParameters, який викликається ззовні.

Сутність LinkedParameters зберігає три приватні значення – два об’єкту, дані яких підлягали обробці та значення розрахованого коефіцієнту та публічні методи для їх зчитування. Безпосередньо розрахунок коефіцієнту винесений в окрему сутність (LinkedParameterCalculation) статичним методом, яка також зберігає список об’єктів типу LinkedParameters .

Оскільки ситуації, в яких показники технологічного процесу виходять з меж норми, скоріше виключення, параметри які будуть оцінюватись на предмет кореляції між собою не мають становити великої кількості, а такий функціонал не підвищить технічні вимоги до встановлення даної системи.[14]

3.2.6 Модуль збереження результатів обробки

Функціонал формування та збереження параметрів розбитий на дві сутності – FormMessage та SaveResults.

Клас FormMessage приймає два набори об’єктів. Перший набір – об’єкти типу CheckedParameter, який був сформований для сутності LinkedParameterCalculation (результати обробки параметрів, які перевищили границю попередження). Другий набір – інкапсульований в самому об’єкті сутності LinkedParameterCalculation – список пар параметрів та сила їх кореляції між собою.

Обидва набори (точніше, список та об’єкт LinkedParameterCalculation) об’явлені за допомогою класу Optional. При формуванні повідомлення спочатку викликається метод, який перевіряє порожні отримані параметри, чи ні. В залежності від цього, викликається приватний метод getSuccessMessage, який формує рядок з поточним часом та текстом «Each parameter is within the normal range», або приватний метод FormErrorMessage, який за допомогою описаного шаблону, формує повідомлення, де вказуються час запису, назва та значення параметру, його статус та всі корелюючи між собою параметри з відповідним коефіцієнтом кореляції.

Клас SaveResults віповідає тільки за роботу з файловою системою. Його основний публічний метод saveResults взаємодіє із внутрішніми, та будує наступну послідовність дій:

- пошук існуючої директорії за заданим шляхом, та в разі її відсутності – створення нової;
- пошук файлу failLog.txt в отриманій директорії, або його створення, в разу відсутності;
- запис в файл нове повідомлення.

В результаті, збережений файл несе собою зручний для обробки людиною текст, який містить в собі детальні результати аналізу даних, які можуть використовуватись для подальшої обробки.

3.3 Алгоритм аналізу даних

Алгоритм аналізу даних розробленої системи зображений у вигляді блок-схеми в ілюстративному матеріалі до даного проекту. Вхідними даними алгоритму виступають набір об’єктів типу Parameter та набір об’єктів типу Setting. Обробка вхідниих даних виконуються згідно наступної послідовності кроків.

Спершу кожен параметр перевіряється на перевищення кожної із заданих границь. Перевірка починається з найбільшої границі – границі аномалії та продовжується послідовний перебір значень границь за спаданням. Якщо значення параметру перевищило визначене для границі значення, це означає, що воно перетнуло і значення нижчих границь і, в такому випаду, всі наступні перевірки пропускаються.

Для кожного параметру в об’єкт типу Calculation, який містить набір попередніх значень параметру, збергається поточне значення.

Після цього, для кожного параметру формується об’єкт типу CheckedParameter, який зберігає результат перевірки (статус параметру), сам параметр, об’єкт типу Calculation та об’єкт типу Setting. Сформовані об’єкти об’єднуються в набір.

Після цього створюється структура даних, в яку зберігається об’єкти, отримані на попередньому кроці, значення параметру в яких перевищило будь-яку границю.

Потім відбувається перевірка, порожня структура або ні. Якщо вона порожня, алгоритм вважається завершеним.

Ящо вона не порожня, перевіряється, знаходиться в ній один елемент, або більше. Якщо один – алгоритм вважається завершеним.

Якщо структура містить більше одного елементу – для кожного з них розраховується середнє арифметичне значення параметру та зберігається в об’єкти типу ClaculationResult.

Після цього об’єкти об’єднуються в пари, за принципом «кожен з кожним»

та зберігаються в об’єктах типу LinkedParameters. Для кожної пари, за наведеною в розділі 3.1 формулою, розраховується коефіцієнт кореляції.

Якщо розрахований коефіцієнт вище 0,3 – об’єкт зберігається в окрему структуру. Після порівняння коефіцієнту кожної пари, алгоритм аналізу даних вважається завершеним, а його результати передаються сутності, яка формує повідомлення, що буде збережено в файлі failLog.txt .

Висновки до розділу 3

В даному розділі була детально описана архітектура розробленої системи, алгоритм її роботи, основні модулі системи, та їх реалізація, наведені приклади роботи системи.Система була спроектована відповідно до поставлених вимог та з використанням визнаних практик, можливостей обраних інструментів розробки та підходящих шаблонів проектування, в на основі цього її архітектуру можна вважати гнучкою до розширення та оптимальною.

Результат роботи системи надається у вигляді зображених графіків на графічному інтерфейсі та збережених результатів аналізу даних. Графічний інтерфейс побудований в мінімалістичному стилі, робота з ним зводиться до простих дій з точки зору користувача.

Відображення оброблених даних створюється в зручній для оператора формі – кожна з характеристик зображена на одному екрані, в разі несправності також на графічний інтерфейс виводиться додаткова інформація. При цьому допоміжні надписи та виділення колькором кожної вагомої деталі допомагають легше орієнтуватись в зображенні та дають можливість зручної та швидкої візуальної оцінки технологічного процесу.

Результати обробки, що зберігаються в текстовому файлі деякою мірою дублюють дані, представлені на графічному інтерфейсі, доповнюючи їх результатами подальшого аналізу та представлені в зручному для обробки людиною форматі.

ВИСНОВКИ

В процесі виконання даного проекту була спроектована та розроблена автоматизована система відображення та аналізу даних технологічного процесу виготовлення шоколаду.

Була детально досліджена предметна область, проведено аналіз існуючих рішень, та на основі цього визначені вимоги до даної системи. Було виявлено, що на сьогоднішній день впровадженого стандарту використання подібних систем на технологічному процесі виготовлення шоколаду не існує. На ринку представлений вибір рішень об’єднує ряд недоліків, серед яких основні – це складність інтеграції системи з технологічним процесом, незрозумілий графічний інтерфейс, висока вартість впровадження системи. Такі недоліки сформували вимоги до розробленої системи, головні з яких – користувацький інтерфейс має бути зручним, простим та зрозумілим на інтуїтивному рівні, а аналіз, який виконує система має бути специфічним та підходящим для технологічного процесу виготовлення шоколаду.

Розроблена система відповідає даним вимогам – графічний інтерфейс максимально лаконічний, використовує засоби привернення уваги до несправності, при цьому доповнений можливістю інтерактивності.[15] Методи аналізу даних в системі також були обрані з огляду на потреби технологічного процесу.

Також, архітектура розробленої системи гнучка до змін та внесення нового функціоналу, тому в разі необхідності, система може бути швидко та легко адаптована відповідно до потреб та рецепту конкретного виробництва.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кокашинский Георгий Романович Производство шоколадных изделий Москва: Пищевая промышленость 1973, 303 ст.
2. Advisory (ICSA-18-338-02) [Електронний ресурс] : Режим доступу: https://ics-cert.us-cert.gov/advisories/ICSA-18-338-02.
3. Simple-scada [Електронний ресурс] : Режим доступу: https://simple-scada.com/
4. Finance pugin [Електронний ресурс] : Режим доступу: https://grafana.com/plugins/ayoungprogrammer-finance-datasource
5. Tiobe index for June 2019 [Електронний ресурс] : Режим доступу: https://www.tiobe.com/tiobe-index/
6. Рейтинг мов програмування 2019 [Електронний ресурс] : Режим доступу: https://dou.ua/lenta/articles/language-rating-jan-2019/
7. Java SE Technologies: Java SE Platform at a Glance [Електронний ресурс] : Режим доступу: https://www.oracle.com/technetwork/java/javase/tech/index-jsp-140763.html
8. A 2D chart library for Java applications [Електронний ресурс] : Режим доступу: https://github.com/jfree/jfreechart
9. Основні фази збирання проекту [Електронний ресурс] : Режим доступу: https://www.apache-maven.ru/lifecycle.html
10. Eclipse Foundetion [Електронний ресурс] : Режим доступу: https://www.eclipse.org/ide/
11. IEEE Standards Association [Електронний ресурс] : Режим доступу: https://standards.ieee.org/standard/1471-2000.html
12. Диспетчер : Моніторинг обладнання [Електронний ресурс] : Режим доступу: https://www.intechnology.ru/monitoringcnc/hardware/
13. Eric Freeman, Elisabeth Robson, Belt Bates, Kathy Sierra Head First Design Patterns O'Reilly Media, Inc 04.11.2004, 656 ст.
14. Gary Pollice, Stanley Selkow, George Heineman Algorithms in a Nutshell O'Reilly Media, Inc 24.10.2009, 364 ст.

15. Nathan Yau Visualize this: The FolowingDataGuide to Design, Visualization, and Statistics 01.06.2011, 358 ст ст.